Scikit-learn

Scikit-learn Library

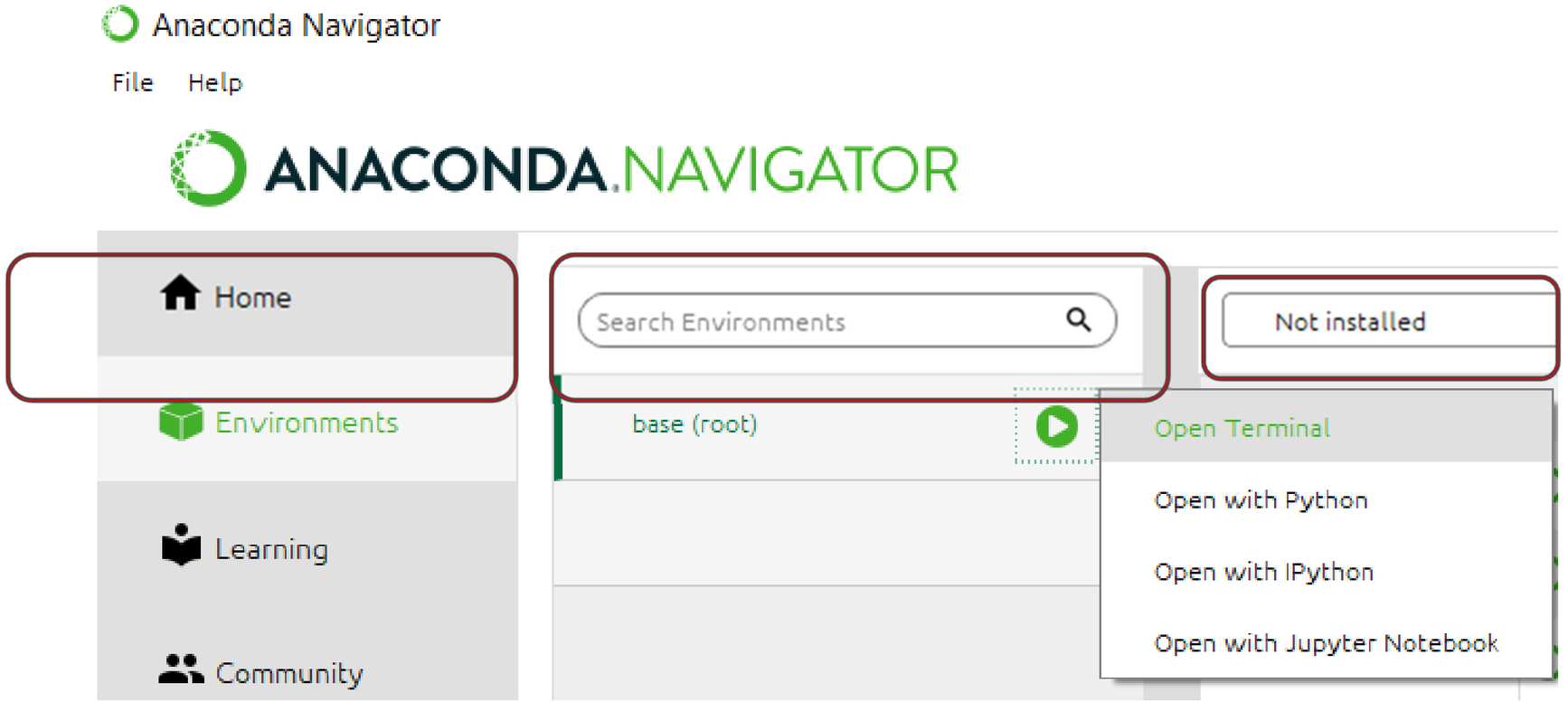
Scikit-learn is a Machine Learning Software Library for the Python programming language

- contains a selections of efficient machine learning algorithms such as
  - Decision Tree, Logistic Regression, K-means

    and many more
- built upon other libraries such as Numpy, Matplotlib and others
  - most of these are already included in the Anaconda package

Aside: Installation of Scikit-learn

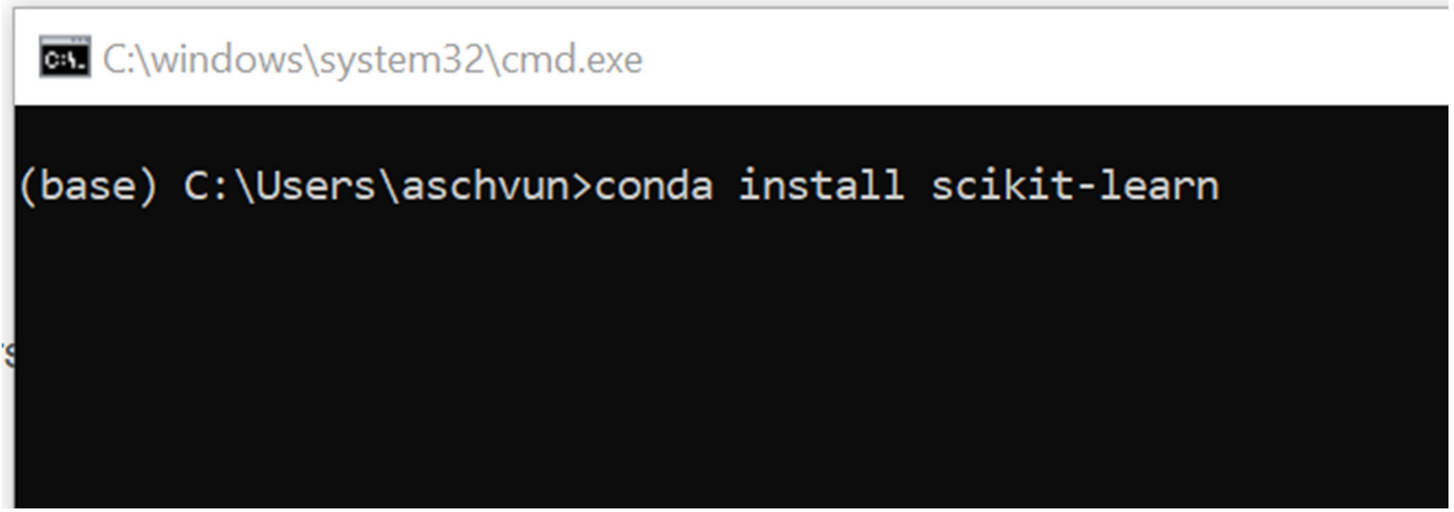
To install the Scikit-Learn using conda

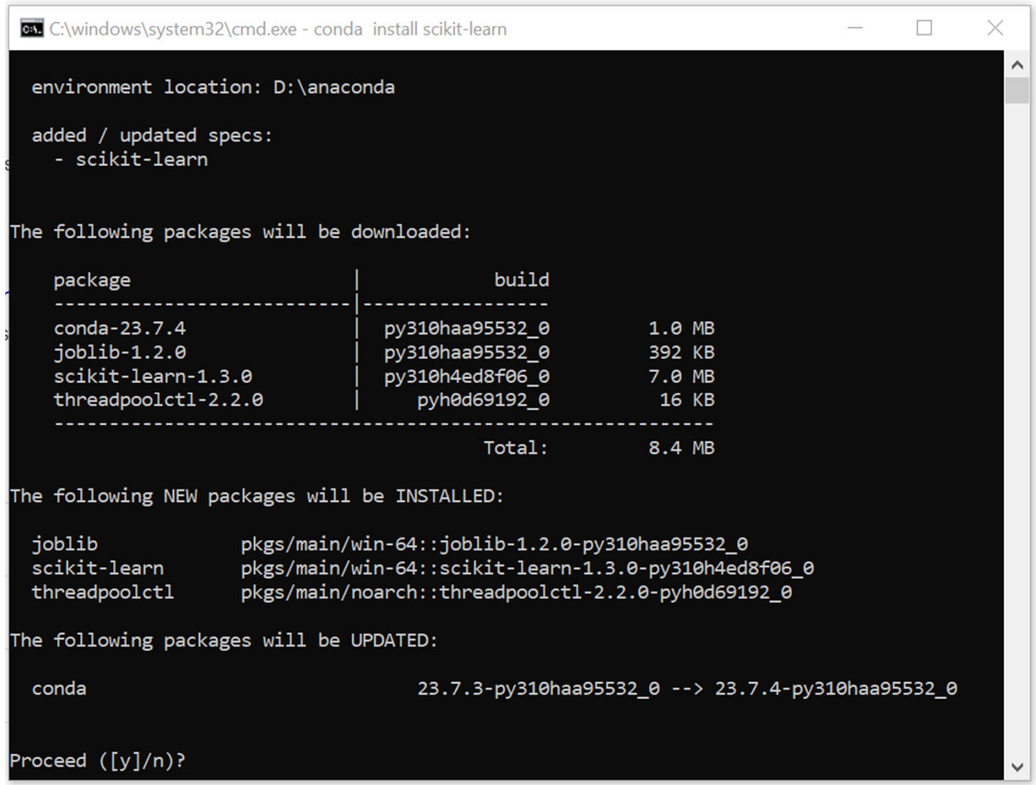


Aside: Installation of Scikit-learn

In the terminal

- type “conda install scikit-learn”





Data Scaling with Scikit-learn

Data Scaling

Data used for machine learning

- contains value of features that may have different ranges and units.
- can affect negatively on the performance of algorithms like KNN, SVM, linear and Logistic Regression.

To address this issue

- scaling is used to standardize data to produce good results

Three common data scaling techniques:

- Standard Scaler
- Min-Max Scaler
- Robust Scaler

Data Scaling Comparisons

Standard Scaler

- scale data to have a mean of 0 and standard deviation of 1
- sensitive to outliers and extreme values
  - skew mean and standard deviation which leads to poor scaling
- not ideal for non-normal distributions

Min-Max Scaler

- scales all data to be in range [0, 1], or in range [-1, 1] if there are negative values
- suitable for non-normal distribution data
- sensitive to outliers and extreme values

Robust Scaler reduces the impact of outliers by

- scaling data using median and interquartile range (IQR) to reduce the impact of outliers
- suitable for data contains many outliers
- may not perform well when data is highly skewed

Data Generation

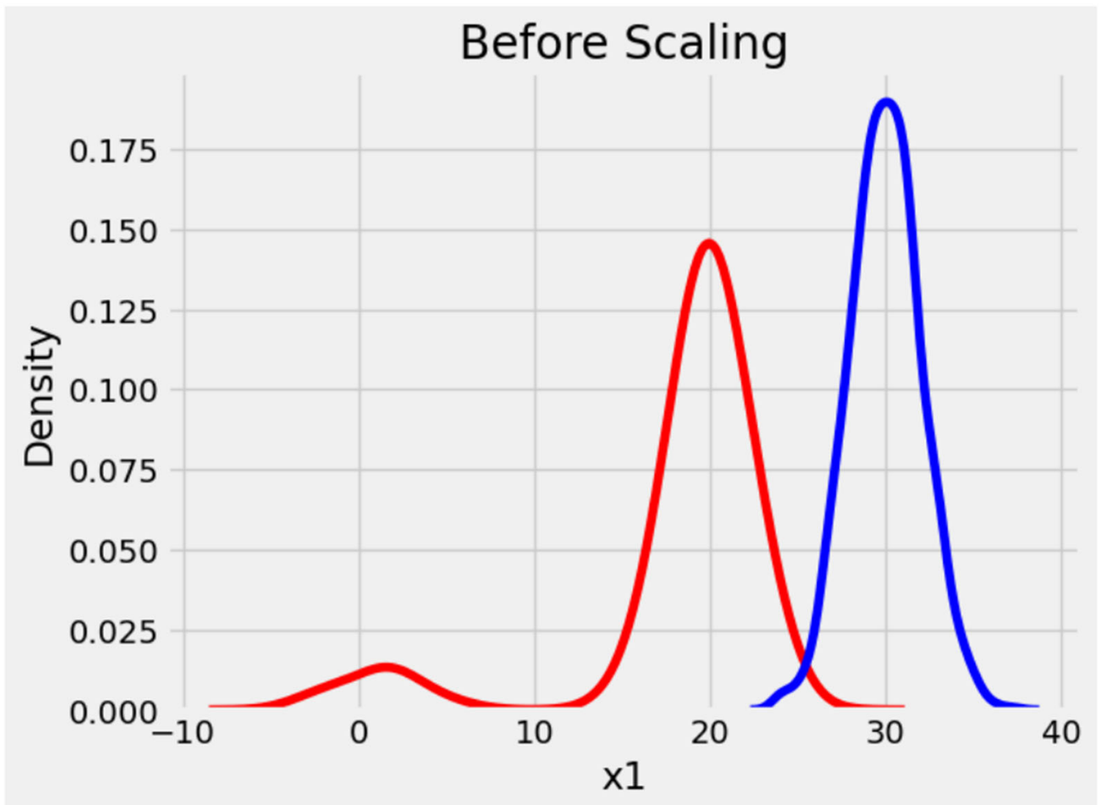
We will first generate two sets of random (normalized) data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
```

```
x = pd.DataFrame({
    'x1': np.concatenate([np.random.normal(20, 2, 1000), np.random.normal(1, 2, 100)]),
    'x2': np.random.normal(30, 2, 1100)  })
#np.random.normal
```

Generation

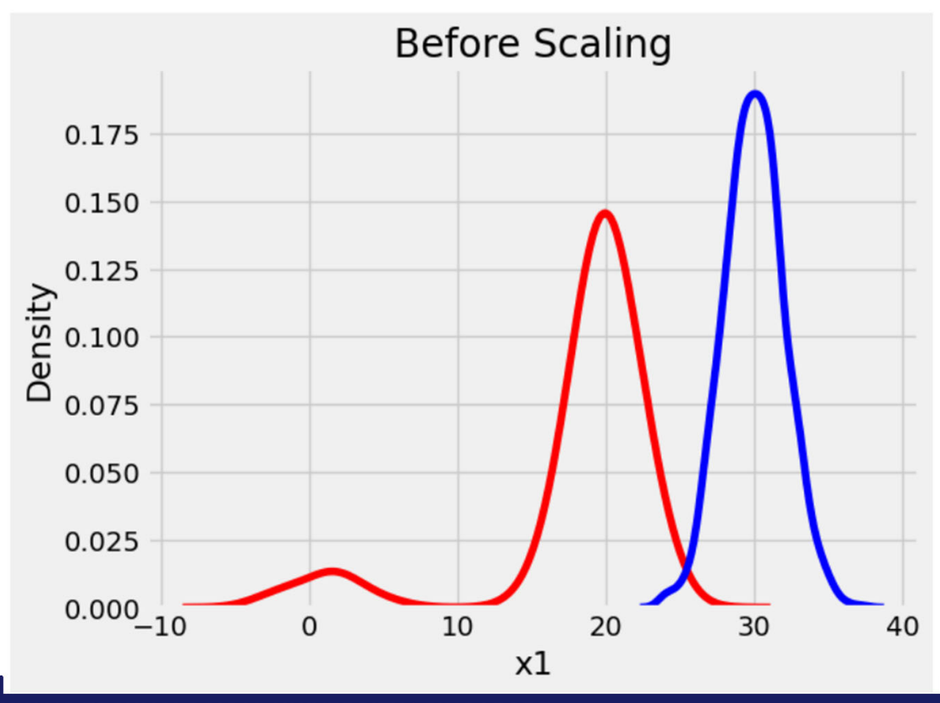
```
plt.title('Before Scaling')
sns.kdeplot(x['x1'], color ='r') # plot its Kernel Density Estimate
sns.kdeplot(x['x2'], color ='b')
plt.show()
```

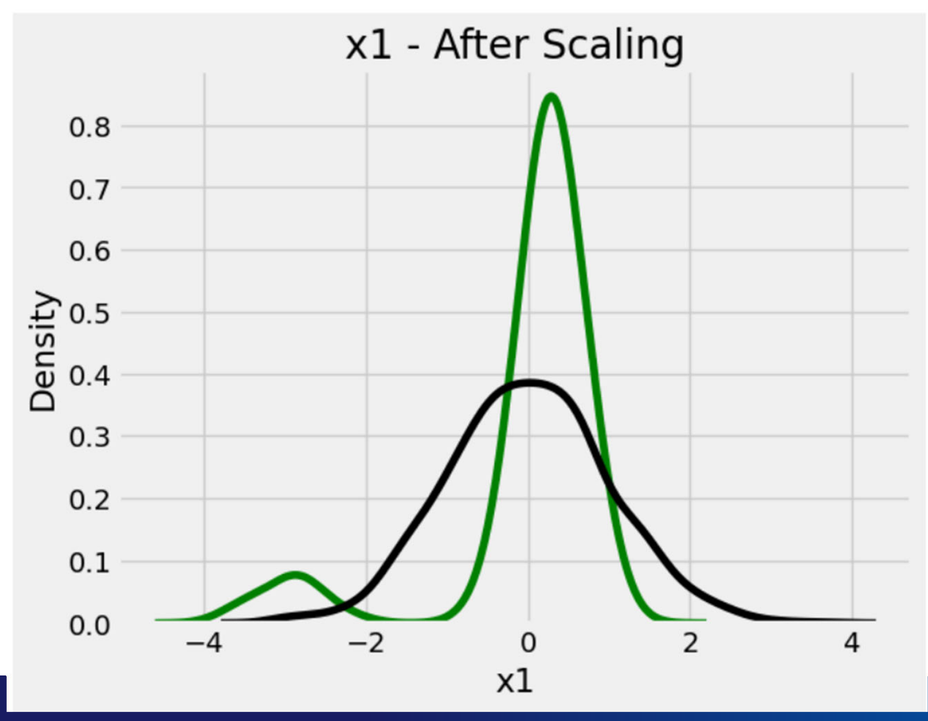


Standard Scaling

```
scaler = preprocessing.StandardScaler() #select the standard scaler (instantiate)
standard_df = scaler.fit_transform(x) #scale the data
standard_df = pd.DataFrame(standard_df, columns =['x1', 'x2'])
```

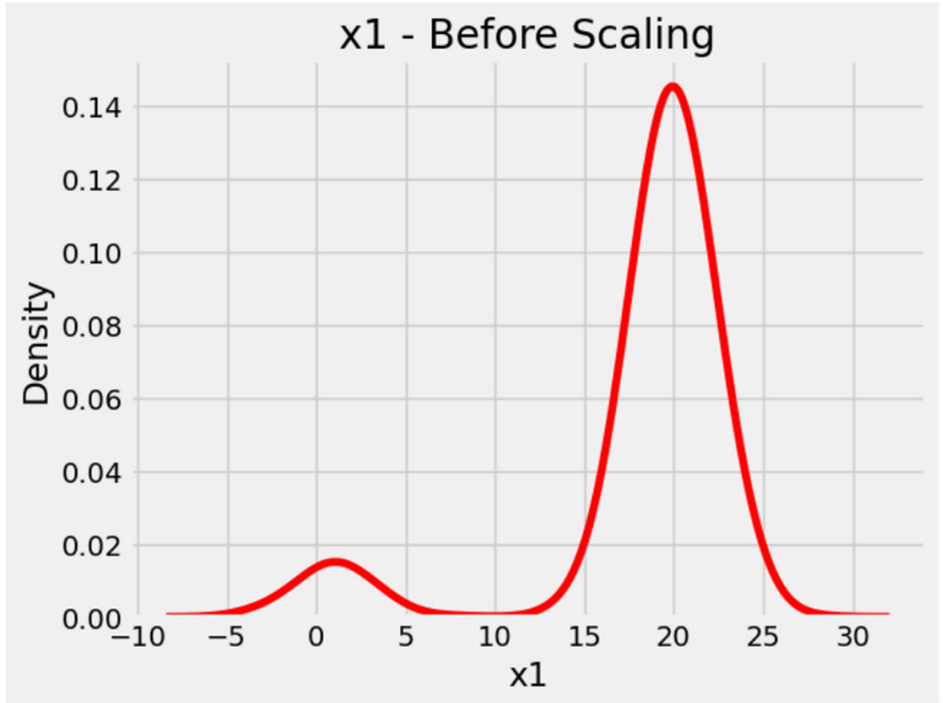
```
plt.title('x1 - After Scaling')
sns.kdeplot(standard_df['x1'], color ='g')
sns.kdeplot(standard_df['x2'], color ='black')
plt.show()
```

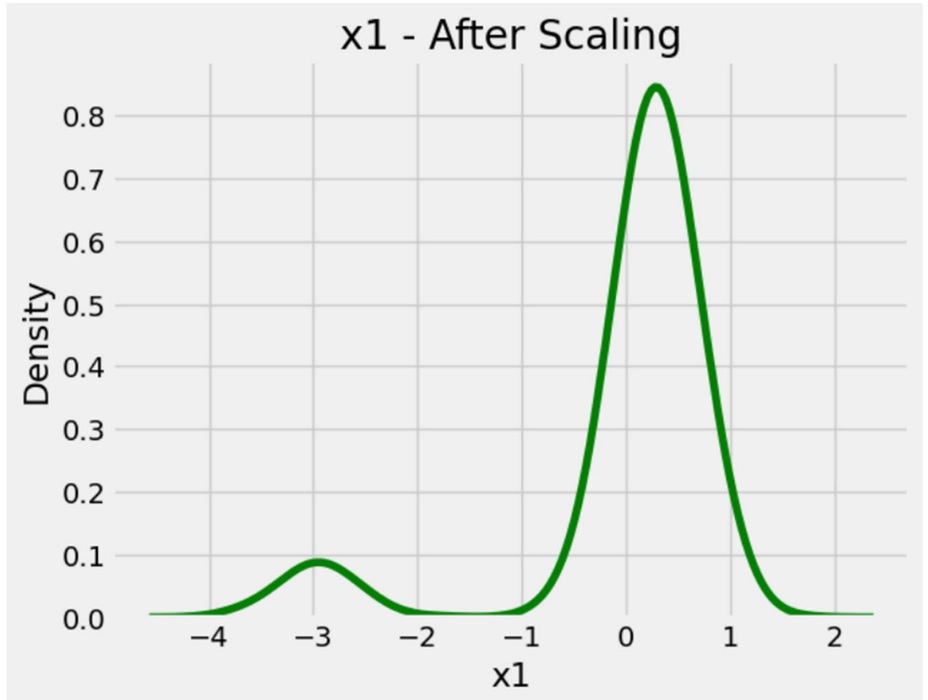




Standard Scaling

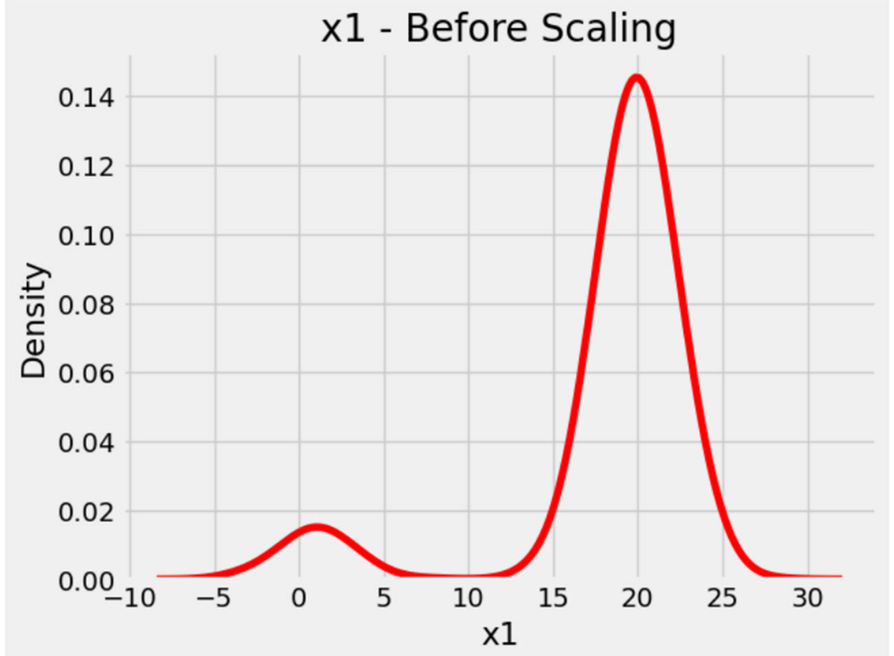
Adjusts data to have a mean of 0 and standard deviation of 1.

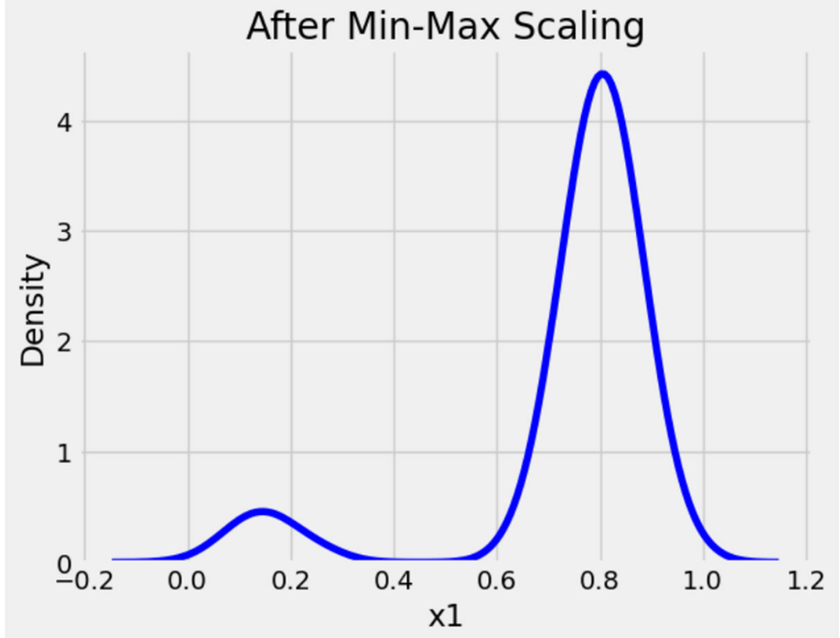




Min-Max Scaling

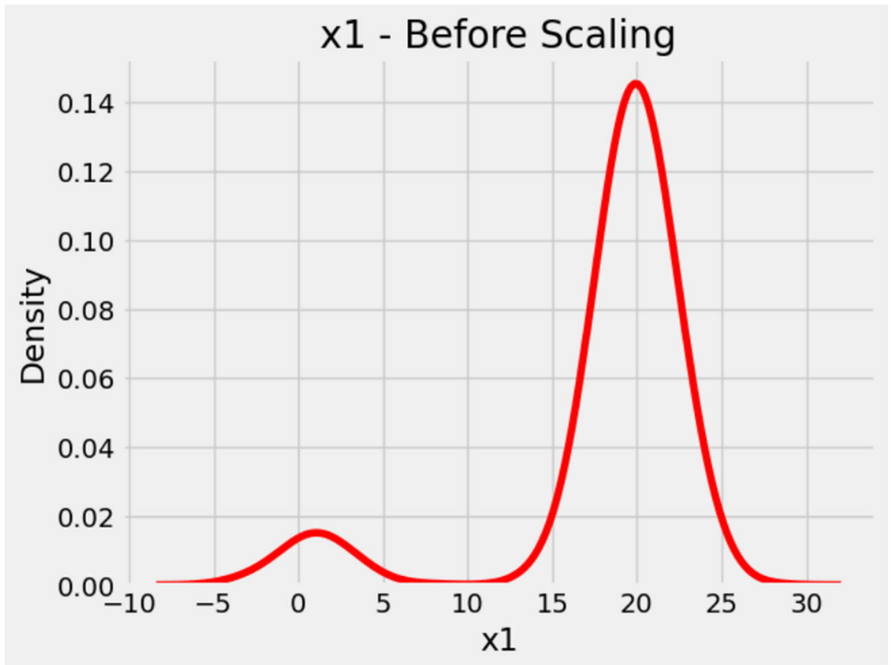
```
scaler = preprocessing.MinMaxScaler()
minmax_df = scaler.fit_transform(x)
minmax_df = pd.DataFrame(minmax_df, columns =['x1', 'x2'])
plt.title('After Min-Max Scaling')
sns.kdeplot(minmax_df['x1'], color ='b')
plt.show()
```

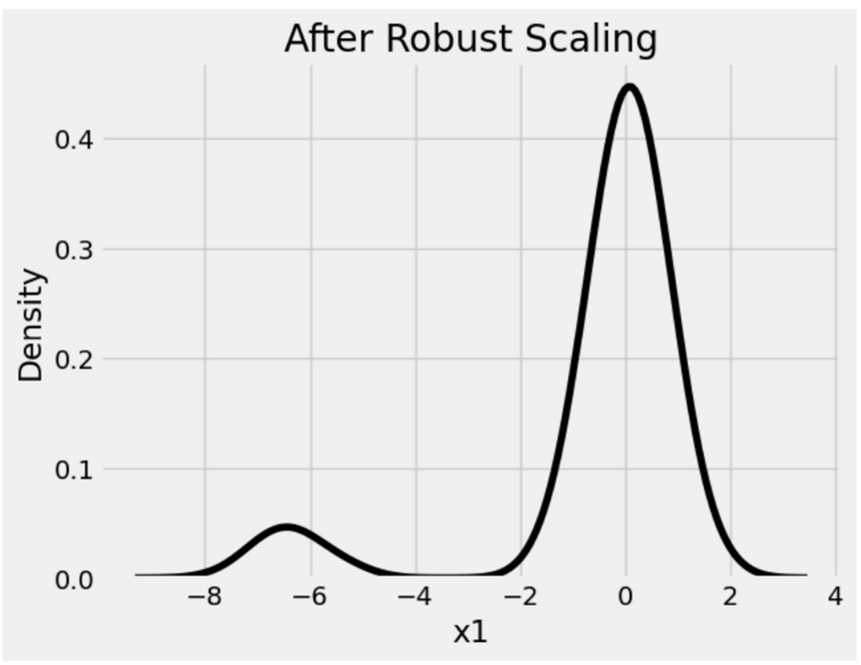




Robust Scaling

```
scaler = preprocessing.RobustScaler()
robust_df = scaler.fit_transform(x)
robust_df = pd.DataFrame(robust_df, columns =['x1', 'x2'])
plt.title('After Robust Scaling')
sns.kdeplot(robust_df['x1'], color ='black')
plt.show()
```

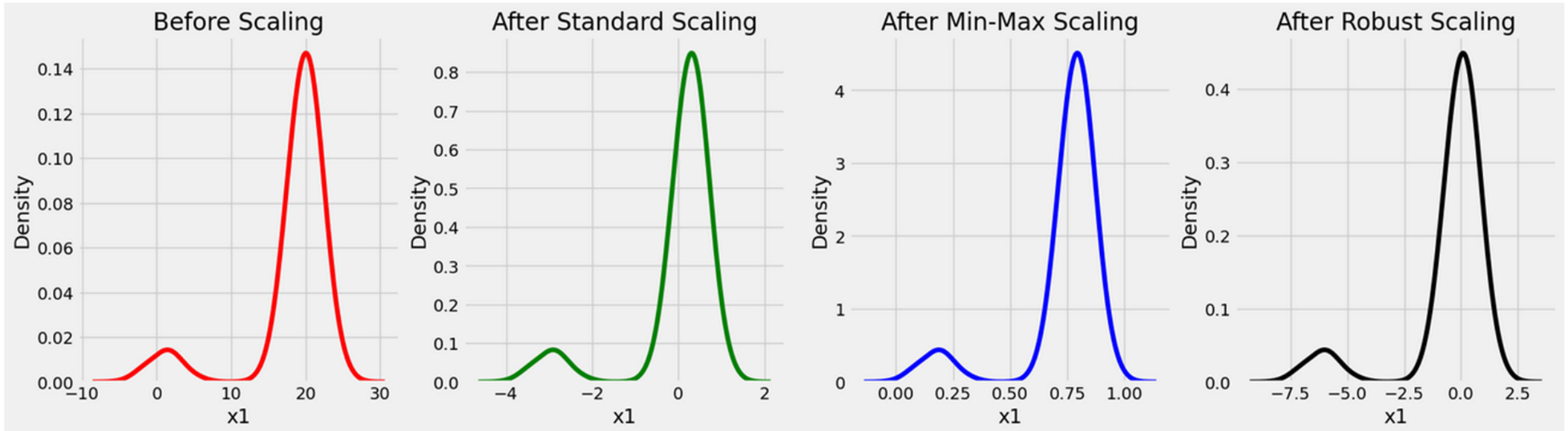




Scaling Comparison

```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(ncols = 4, figsize =(20, 5))

ax1.set_title('Before Scaling')
sns.kdeplot(x['x1'], ax = ax1, color ='r')
```



ML with Scikit-learn

Scikit-learn based ML

This is a basic example of training a model using Scikit-Learn to recommend accommodation based on budget (data in Rental.csv).

```
Condo,Rental,Room
1,1000,1
1,1200,1
1,1500,1
1,2000,2
1,2400,2
1,2700,2
1,3300,2
1,3500,3
1,3800,3
1,4000,3
1,4500,3
1,4800,4
1,5000,4
1,5500,4
2,800,1
2,1000,1
2,1300,1
2,1500,2
2,2000,2
2,2200,2
2,3000,2
2,3300,3
2,3600,3
2,3800,3
2,4300,3
2,4600,4
2,4800,4
2,5300,4
```

Scikit-learn based ML

First Create a DataFrame from Rental.csv contents

```
import pandas as pd
rental_data = pd.read_csv('rental.csv')
rental_data
```

	Condo	Rental	Room
0	1	1000	1
1	1	1200	1
2	1	1500	1
3	1	2000	2
4	1	2400	2
5	1	2700	2
6	1	3300	2
7	1	3500	3
8	1	3800	3
9	1	4000	3

Scikit-learn based ML

Next, we separate the data set into two groups

i. Input: X (used to train the model)

ii. Output: y (the actual values to check against model's predictions)

```
#split the dataset into 2 sets - input set X (Condo Rental) and output set y
X=rental_data.drop(columns=['Room'])
X
```

```
y = rental_data['Room']
y
```

Scikit-learn based ML

The dataset is separated into two groups

i. Input: X
ii. Output: y

x

	Condo	Rental
0	1	1000
1	1	1200
2	1	1500
3	1	2000
4	1	2400
5	1	2700

y

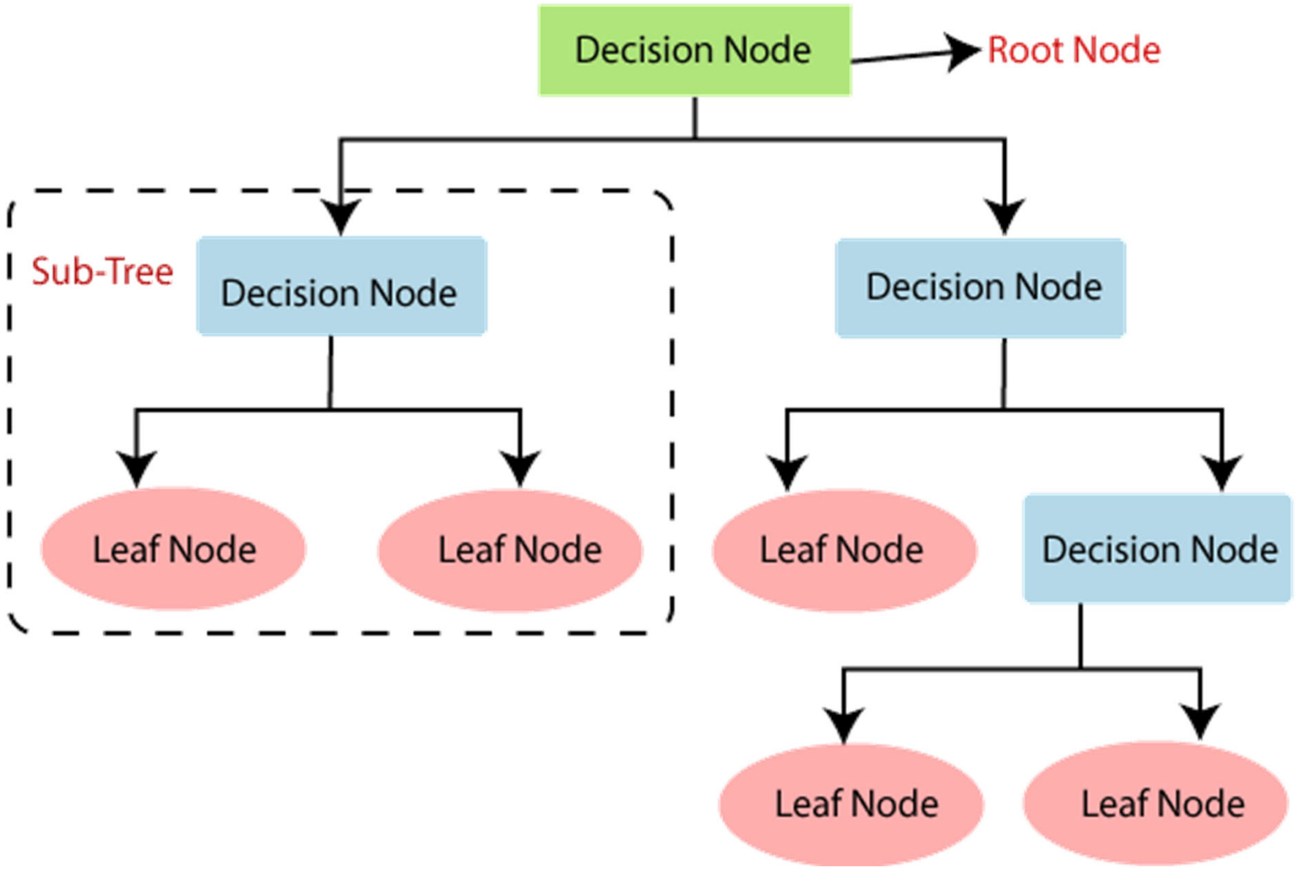
```
0    1
1    1
2    1
3    2
4    2
5    2
```

	Condo	Rental	Room
0	1	1000	1
1	1	1200	1
2	1	1500	1
3	1	2000	2
4	1	2400	2
5	1	2700	2

Scikit-learn based ML

We will select the Decision Tree Classifier algorithm to train our model

- Decision Tree is a Supervised Learning technique



Training a Model

Import the Decision Tree Classifier class of algorithm from SciKit-learn

```
from sklearn.tree import DecisionTreeClassifier  # use this class of algorithm
```

Create a new instance of this class

```
model=DecisionTreeClassifier()  # create a new instance of this claas
```

Train the model: fit()

```
model.fit(X.values,y.values) # train the model
```

Model Prediction

Budget of $1450

```
#type of room avaiable in COndo 1 and condo 2 with budget 1450
predictions=model.predict([[1,1450],[2,1450]])
```

Predicted output

```
predictions #the room type from Conda 1 and 2
```

```
array([1, 2], dtype=int64)
```

Does this look correct?

- Condo 1 = 1-room type
- Condo 2 = 2-room type

Try some other values

```
1  •Condo,Rental,Room
2  1,1000,1
3  1,1200,1
4  1,1500,1
5  1,2000,2
6  1,2400,2
7  1,2700,2
8  1,3300,2
9  1,3500,3
10 1,3800,3
11 1,4000,3
12 1,4500,3
13 1,4800,4
14 1,5000,4
15 1,5500,4
16 2,800,1
17 2,1000,1
18 2,1300,1
19 2,1500,2
20 2,2000,2
21 2,2200,2
22 2,3000,2
23 2,3300,3
24 2,3600,3
25 2,3800,3
26 2,4300,3
27 2,4600,4
28 2,4800,4
29 2,5300,4
```

Model Training – Training and Testing

In machine learning training of model

- the data set will usually be separated into two or three sets
  - Training, Testing and Validation
    - for small data set, Validation set is usually omitted

Split the dataset into two sets:Train_test_split()

```
#To measure accuracy of model 80% for traning, 20% for testing
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

Model Training – Training and Testing

Training the model using the training set

```
model.fit(X_train.values,y_train.values) # train the model
```

Run the model using the testing set and get its predicted value

```
predictions = model.predict(X_test.values)
```

Check against the y_test ‘truth’ values and output the accuracy

```
#compare the predcted value with y_test
from sklearn.metrics import accuracy_score
accuracy_score(y_test, predictions)
```

```
Out[35]: 0.8333333333333334
```

Model Training – Training and Testing

To get a better measurement of accuracy of the model

- we need to do more rounds of testing
  - using different sets of test data

But we only have small data set

- we will re-generate the Training & Testing sets, and run them multiple times to observe the variations in the accuracy score

```
#run mutliple times the test_split cycles,
#answers will change because different test sets are used by test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
model.fit(X_train.values,y_train.values) # train the model
predictions = model.predict(X_test.values)
accuracy_score(y_test, predictions)
# To re-run the above code, use "ctrl-Enter"
```

- how is the score like?

Saving the Model

Once we are satisfied with the model performance

- we will store the model to a file
  - and use it for inference later

Use the joblib software toolset to save model

```
import joblib  #tools to save model
```

```
joblib.dump(model,'rental-budget.joblib')  # save the model into a file
```

Running Inference

To run inference using the saved model

- start a new notebook
- load the model
- run the prediction

```
In [1]: import joblib

In [2]: model_loaded=joblib.load('rental-budget.joblib')

In [3]: predictions = model_loaded.predict([[1,3000],[2,3000]])

In [4]: predictions

Out[4]: array([2, 2], dtype=int64)
```

Iris dataset with Scikit-learn

“Hello World” of ML – Iris dataset

Widely used as a beginner's dataset for machine learning

- 3 types of iris flowers – Versicolor, Setosa, Virginica
- Total of 150 samples - 50 samples for each iris type
- 4 attributes of each flowers - length and the width of the sepal and petal

Iris Versicolor **Iris Setosa** **Iris Virginica**

Iris dataset

Iris dataset is part of Scikit-learn library

Loading the Iris dataset

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.datasets import load_iris
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
iris = load_iris()
```

Iris dataset

Load it into DataFrame and examine the contents

```
df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['species'] = iris.target #set the target as species
X = iris.data
df.head(5) #see the first 5 samples
#note thtat there are 4 features, Sepal length/Width and Petal length/Width
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Iris dataset

Load it into DataFrame and examine the contents

```
df.sample(4) #print 4 random samples
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
146	6.3	2.5	5.0	1.9	2
90	5.5	2.6	4.4	1.2	1
54	6.5	2.8	4.6	1.5	1
16	5.4	3.9	1.3	0.4	0

```
df.sample(frac=0.02)    #print fraction of the sample
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
110	6.5	3.2	5.1	2.0	2
117	7.7	3.8	6.7	2.2	2
105	7.6	3.0	6.6	2.1	2

Iris dataset

Examining the contents

```
df.describe() #print a numerical summary of each attribute using describe method
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

Iris dataset

Examining contents

```
df.groupby('species').size()
```

```
species
0     50
1     50
2     50
dtype: int64
```

```
# Show the target names
species_names = iris.target_names
print(species_names)
```

```
['setosa' 'versicolor' 'virginica']
```

Model Training

Prepare the data set for training

Split into the input/output groups

```
# Prepare training data for building the model
X_train = df.drop(['species'], axis=1)
y_train = df['species']
```

Use the Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
```

```
# Instantiate the model
model = DecisionTreeClassifier()
```

```
# Train/Fit the model
model.fit(X_train.values, y_train.values)
```

Testing of Model

Do a quick test

```
# Make prediction using the model
X_pred = [5.1, 3.2, 1.5, 0.5]
y_pred = model.predict([X_pred])
```

```
y_pred
```

```
]: array([0])
```

```
print("Prediction is: {}".format(species_names[y_pred]))
```

```
Prediction is: ['setosa']
```

Model with different Algorithms

To use a different algorithm

```
#Logistic regression is a linear classifier.
from sklearn.linear_model import LogisticRegression
# Instantiate the model
model = LogisticRegression(solver='lbfgs', max_iter=400)  #increase iteration to
model.fit(X_train.values, y_train.values)
X_pred = [5.1, 3.2, 1.5, 0.5]
y_pred = model.predict([X_pred])
```

```
print("Prediction is: {}".format(species_names[y_pred]))
```

```
Prediction is: ['setosa']
```

Model with different Algorithms

Another algorithm

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(X_train.values, y_train.values)
X_pred = [5.1, 3.2, 1.5, 0.5]
y_pred = model.predict([X_pred])
```

```
print("Prediction is: {}".format(species_names[y_pred]))
```

```
Prediction is: ['setosa']
```

Model with different Algorithms

Yet another algorithm

```
# used a adaBoostClassifier
from sklearn.ensemble import AdaBoostClassifier
# Instantiate the model
model = AdaBoostClassifier()
model.fit(X_train.values, y_train.values)
X_pred = [5.1, 3.2, 1.5, 0.5]
y_pred = model.predict([X_pred])
```

```
print("Prediction is: {}".format(species_names[y_pred]))
```

```
Prediction is: ['setosa']
```