Machine Learning- Predicitve Modelling & Data Transformation

Estimated time needed: **40** minutes

Objectives

After completing this lab you will have a good understanding in:

- Train-Test and Validation Set
- Data Preprocessing methods - Standardization & Normalization
- Metrics for Evaluation
- Need for Predicitve Modelling

Libraries Focused

- Sklearn
- Pandas
- NumPy
- Matplotlib, Seaborn
- Statsmodels

This Notebook is created for Python Module

How we define Machine Learning ?

Machine learning is the process of programming computers to optimize a performance criterion using example data or past experience. We define a model up to some parameters and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model developed may be **predictive** to make predictions in the future, or **descriptive** to gain knowledge from data, or both.

We would be focusing on the **Predicitve Modelling Techniques**

UC Berkeley breaks out the learning system of a machine learning algorithm into three main parts.

- A Decision Process
- An Error Function
- An Model Optimization

1.What is Predictive Modelling ?

Predictive analytics is a branch of advanced analytics that makes predictions about future outcomes using historical data combined with statistical modeling, data mining techniques and machine learning. Companies employ predictive analytics to find patterns in this data to identify risks and opportunities.To gain insights from this data, data scientists use deep learning and machine learning algorithms to find patterns and make predictions about future events.

Use Case in Industries

- **Banking**

Financial services use machine learning and quantitative tools to predict credit risk and detect fraud.

- **Healthcare**

Predictive analytics in health care is used to detect and manage the care of chronically ill patients.

- **Human resources (HR)**

HR teams use predictive analytics to identify and hire employees, determine labor markets and predict an employee's performance level.

- **Marketing and sales**

Predictive analytics can be used for marketing campaigns throughout the customer lifecycle and in cross-sell strategies.

- **Retail**

Retailers use predictive analytics to identify product recommendations, forecast sales, analyze markets and manage seasonal

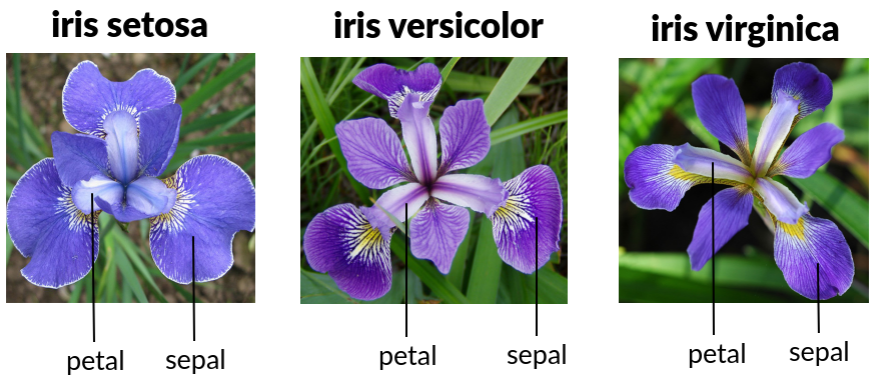
inventory.

- **Supply chain**

Businesses use predictive analytics to make inventory management more efficient, helping to meet demand while minimizing stock.

Loading Data from Sklearn

We would be using IRIS DATASET from Scikit-learn, We will see a Linear Regression Model and also will develop a Function for LR. Before that,Let's know more on the dataset.



Data Set Characteristics:

```
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class( Output ):
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988
```

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper.The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter

In [1]:
```
from sklearn.datasets import load_iris
iris = load_iris() # load iris dataset to iris variable
```

Features – The variables of data are called its features. They are also known as predictors, inputs or attributes.

1. **Feature matrix** – It is the collection of features, in case there are more than one., cols on basis of what we predict output
2. **Feature Names** – It is the list of all the names of the features.

In [2]:
```
feature_matrix = iris.data
feature_names = iris.feature_names
```

In [3]:
```
feature_names
```

Out[3]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [4]:
```
feature_matrix[:5] # printing first 5 values
```

Out[4]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

Response – It is the output variable that basically depends upon the feature variables. They are also known as target, label or output.

1. **Response Vector** – It is used to represent response column. Generally, we have just one response column.
2. **Target Names** – It represent the possible values taken by a response vector.

In [5]:
```
target_matrix = iris.target
target_name   = iris.target_names
```

In [6]:
```
target_matrix
```

Out[6]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [7]:
```
target_name
```

Out[7]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [8]:
```
# Let's Seperate the Feature and Target as X and y
X = iris.data # feature_matrix
y = iris.target # target_matrix
```

In [9]:
```
# Load Libraries
import pandas as pd
import numpy as np
import sklearn
import statistics
import statsmodels
import seaborn as sns
import matplotlib.pyplot as plt
```

In [10]:
```
df = pd.DataFrame(feature_matrix, columns=feature_names)
# add target column to df
df['species (target)'] = target_matrix
df.sample(3)
```

Out[10]:

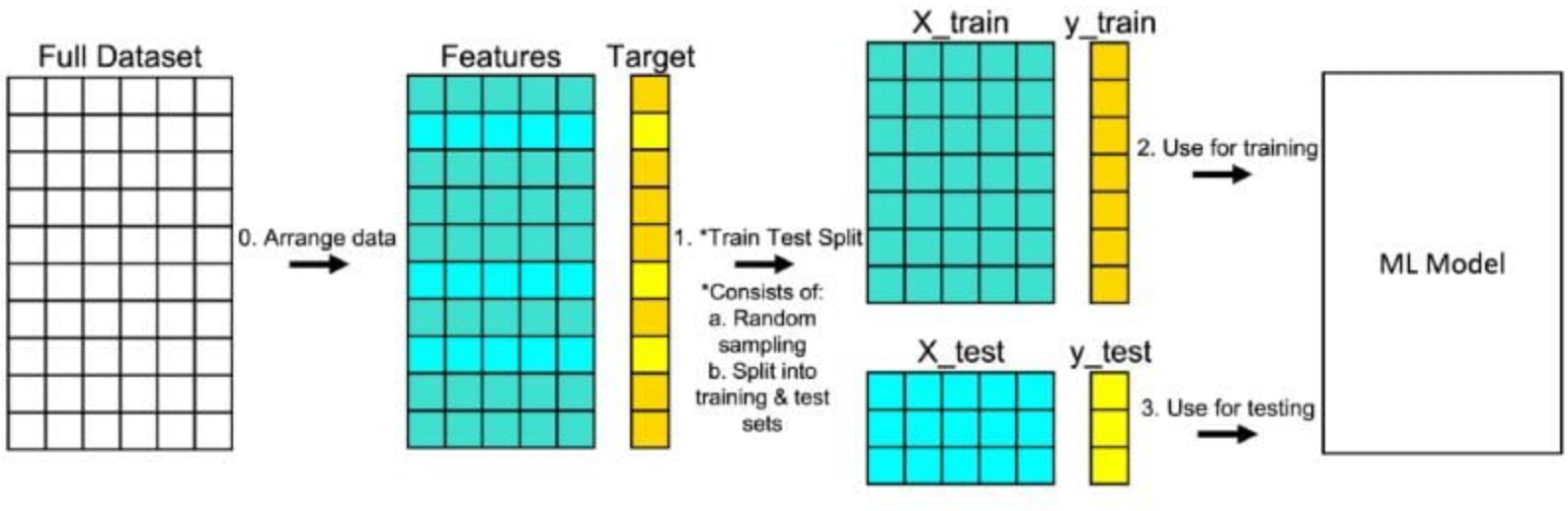
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species (target)
105	7.6	3.0	6.6	2.1	2
102	7.1	3.0	5.9	2.1	2
27	5.2	3.5	1.5	0.2	0

In [11]:
```
# Function to replace target matrix with target names
def replace_target(x):
    if x == 0:
        return 'setosa'
    elif x == 1:
        return 'versicolor'
    elif x == 2:
        return 'virginica'

# Now let's create a new column by applying *replace_target* function to sepcies(target) col
df['species-names'] = df['species (target)'].apply(replace_target)
```

In [12]:
```
df.sample(4)
```

Out[12]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species (target)	species-names
128	6.4	2.8	5.6	2.1	2	virginica
1	4.9	3.0	1.4	0.2	0	setosa
6	4.6	3.4	1.4	0.3	0	setosa
10	5.4	3.7	1.5	0.2	0	setosa



Preparing Data for Modelling

- Training Dataset : Using the training dataset a model is trained or build.
- Testing Data set : Portion of data which is not used in training nor in evaluation part
- Validation data set: Accurate dataset of the model on the useen dataset

By using Train test Split we can seperate the training and testing dataset

```
from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test(X, y, train/test_size, random_state)
* X_train : the matrix used for training
* X_test : used for predictions
* y_train: output for the matrix X_train for trainning model
* y_test : evaluation will be made with y_test and predictions
* random_state : defines the shuffling number before splitting the data
* train/test_size: defines the percentage you want to used for training/testing [use in 0.2 - 0.9]
```

In [13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.6, random_state = 101)
X_train.shape, X_test.shape
```

Out[13]:
```
((90, 4), (60, 4))
```

Linear Regression

$y = mx + C$

In [14]:
```
from sklearn.linear_model import LinearRegression
lin_model = LinearRegression()
lin_model.fit(X_train, y_train)

# making predictions
predictions = lin_model.predict(X_test)
```

Regression Evaluation Metrics

After Modelling we need to check the accuracy or the loss values of the model. So, Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

In [15]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
print("Mean absolute error: %.2f" % mean_absolute_error(predictions,y_test))
print("Residual sum of squares (MSE): %.2f" % mean_squared_error(predictions,y_test))
print("R2-score: %.2f" % r2_score(predictions , y_test))
```

```
Mean absolute error: 0.17
Residual sum of squares (MSE): 0.05
R2-score: 0.91
```

Let's make a Function to LR

```
def Linear_Model_Generator(X, y, tr_size):
    from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = tr_size, random_state =
101)
    print('X_train: ', X_train.shape)
    print('X_test: ', X_test.shape)

    print('-----------Building Model------------')
    from sklearn.linear_model import LinearRegression
    model = LinearRegression()
    model.fit(X_train, y_train)

    # fit vlues for xtest
    pred = model.predict(X_test)

    print("Mean absolute error: %.2f" % mean_absolute_error(pred,y_test))
    print("Residual sum of squares (MSE): %.2f" % mean_squared_error(pred,y_test))
    print("Root Residual sum of squares (RMSE): %.2f" % np.sqrt(mean_squared_error(pred,y_test)))
    print("R2-score: %.2f" % r2_score(pred , y_test))
```

In [16]:
```
def Linear_Model_Generator(X, y, tr_size):
    from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = tr_size, random_state = 101)
    print('X_train: ', X_train.shape)
    print('X_test: ', X_test.shape)

    print('-----------Building Model------------')
    from sklearn.linear_model import LinearRegression
    model = LinearRegression()
    model.fit(X_train, y_train)

    # fit vlues for xtest
    pred = model.predict(X_test)

    print("Mean absolute error: %.2f" % mean_absolute_error(pred,y_test))
    print("Residual sum of squares (MSE): %.2f" % mean_squared_error(pred,y_test))
    print("Root Residual sum of squares (RMSE): %.2f" % np.sqrt(mean_squared_error(pred,y_test)))
    print("R2-score: %.2f" % r2_score(pred , y_test))
```

In [17]:
```
# Performing Test on the function made
data = pd.read_csv('FuelConsumption.csv')
X = data[['ENGINESIZE']] #data.drop('CO2EMISSIONS', axis = 1)
y = data['CO2EMISSIONS']
Linear_Model_Generator(X,y, tr_size=0.3)
```

```
X_train:  (746, 1)
X_test:  (321, 1)
-----------Building Model------------
Mean absolute error: 22.91
Residual sum of squares (MSE): 886.73
Root Residual sum of squares (RMSE): 29.78
R2-score: 0.73
```

In [18]:
```
sns.regplot(X,y)
plt.show()
```

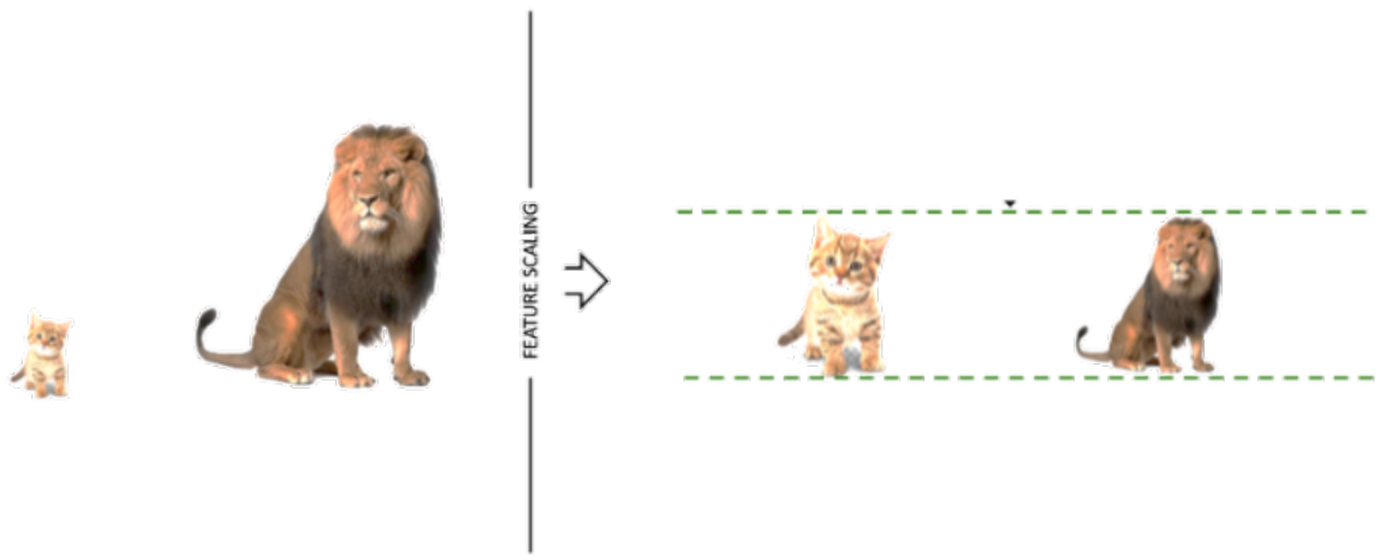
```
/Users/sumitkumarshukla/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument wi
ll be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpre
tation.
  warnings.warn(
```

2. Data Transformations

Data transformation is an intermediate step for transforming the data set to be analyzed and to make it more suitable for subsequent analytical processing. The transformation changes values of selected attributes to satisfy requirements for the algorithms that are used for predictive modeling, for example, classification, regression, clustering, or association rule mining.

Why we need to Scale Data

Scaling of the data comes under the set of steps of data pre-processing when we are performing machine learning algorithms in the data set. Scaling the target value is a good idea in regression modelling; scaling of the data makes it easy for a model to learn and understand the problem.



As we know, most of the machine learning models learn from the data by the time the learning model maps the data points from input to output. And the distribution of the data points can be different for every feature of the data. Larger differences between the data points of input variables increase the uncertainty in the results of the model.

The machine learning models provide weights to the input variables according to their data points and inferences for output. In that case, if the difference between the data points is so high, the model will need to provide the larger weight to the points and in final results, the model with a large weight value is often unstable. This means the model can produce poor results or can perform poorly during learning.

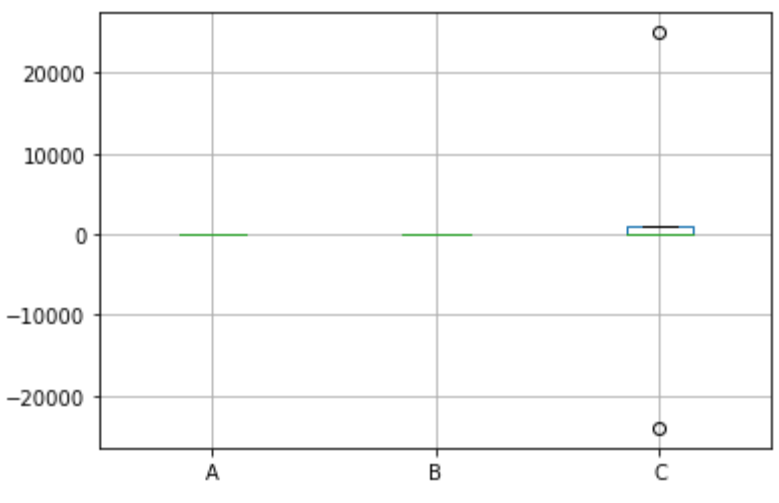
In [1]:
```
import pandas as pd
d = {'A': [1, 2, -0.5, 4, 3],
     "B": [100, -25, 0, 94, -24],
     'C': [1005, 25000, -0.3, -23987, 3.5]}
df = pd.DataFrame(d)
df
```

Out[1]:

	A	B	C
0	1.0	100	1005.0
1	2.0	-25	25000.0
2	-0.5	0	-0.3
3	4.0	94	-23987.0
4	3.0	-24	3.5

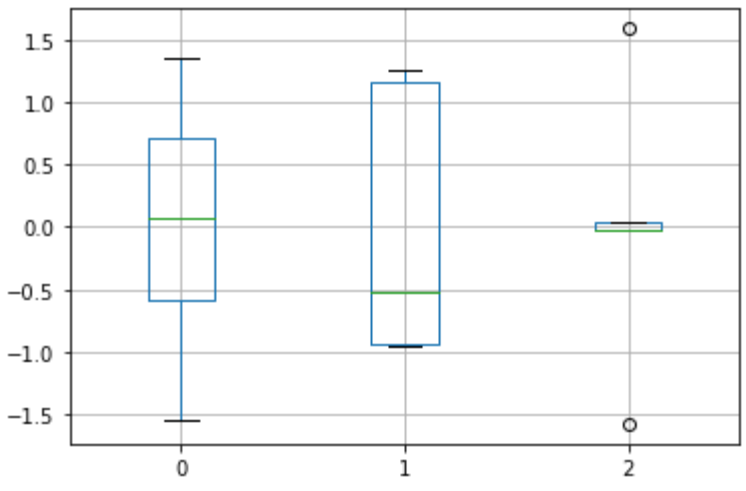
In [3]:
```
df.boxplot()
```

Out[3]: <AxesSubplot:>



In [128…

```
# GLANCE AFTER SCALING
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(df)
scal_df = scaler.transform(df)
sdf = pd.DataFrame(scal_df)
sdf.boxplot()
plt.show()
```



Normalization

Normalization can have various meanings, in the simplest case normalization means adjusting all the values measured in the different scales, in a common scale. Normalization is the method of rescaling data where we try to fit all the data points between the range of 0 to 1 so that the data points can become closer to each other.

It is a very common approach to scaling the data. In this method of scaling the data, the minimum value of any feature gets converted into 0 and the maximum value of the feature gets converted into 1.

Basically, under the operation of normalization, the difference between any value and the minimum value gets divided by the difference of the maximum and minimum values. We can represent the normalization as follows.

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Where x is any value from the feature x and min(X) is the minimum value from the feature and max(x) is the maximum value of the feature. Scikit learn provides the implementation of normalization in a preprocessing package. Let's see how it works.

Implementing Min-Max Scaler

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
```

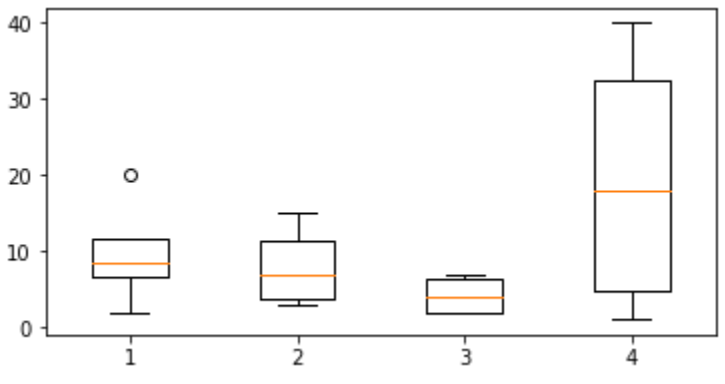
In [43]:

```
x=np.array([[2, 3, 7, 30],
                [9, 4, 6, 1],
                [8, 15, 2, 40],
                [20, 10, 2, 6]])
print(x)
```

```
[[ 2  3  7 30]
 [ 9  4  6  1]
 [ 8 15  2 40]
 [20 10  2  6]]
```

In [47]:

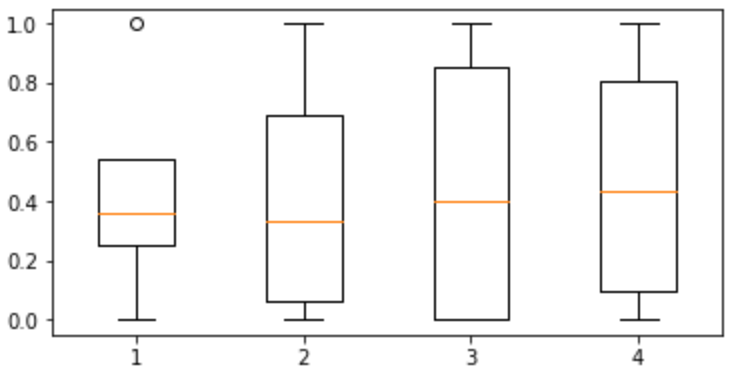
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize =(6, 3))
plt.boxplot(df)
plt.show()
```



In [49]:
```
# Normalizing data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler().fit(df)
scaled_features = scaler.transform(df)
print(scaled_features)
```

```
[[0.         0.         1.         0.74358974]
 [0.38888889 0.08333333 0.8        0.        ]
 [0.33333333 1.         0.         1.        ]
 [1.         0.58333333 0.         0.12820513]]
```

In [50]:
```
fig = plt.figure(figsize =(6, 3))
plt.boxplot(scaled_features)
plt.show()
```



Where to use Normalization ?

Since we have seen the normalization method scales between zero to one it is better to use with the data where the distribution of the data is not following the Gaussian distribution or we can apply with an algorithm that does not count on the distribution of the data in the procedure like K-means and KNN.

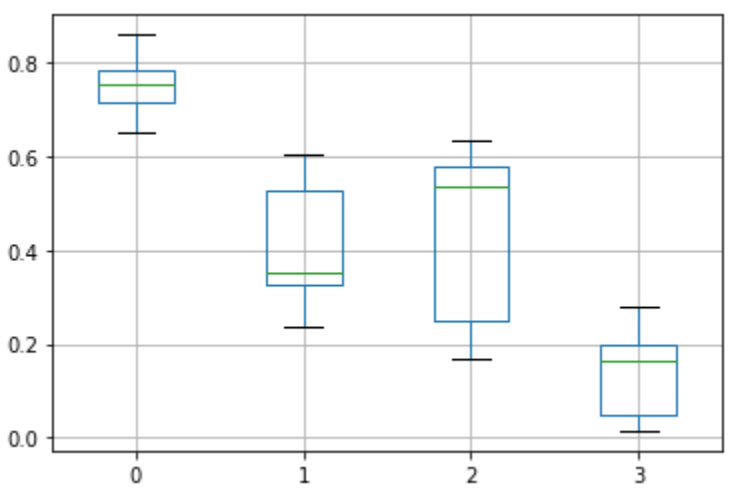
In [59]:
```
mini = np.min(X)
maxi = np.max(X)

sample_data = df['sepal length (cm)'].sample(5)

ser = (df['sepal length (cm)'] - mini )/(maxi - mini)
ser.loc[sample_data.index.values.tolist()]
```

Out[59]:
```
149    0.743590
97     0.782051
44     0.641026
54     0.820513
90     0.692308
Name: sepal length (cm), dtype: float64
```

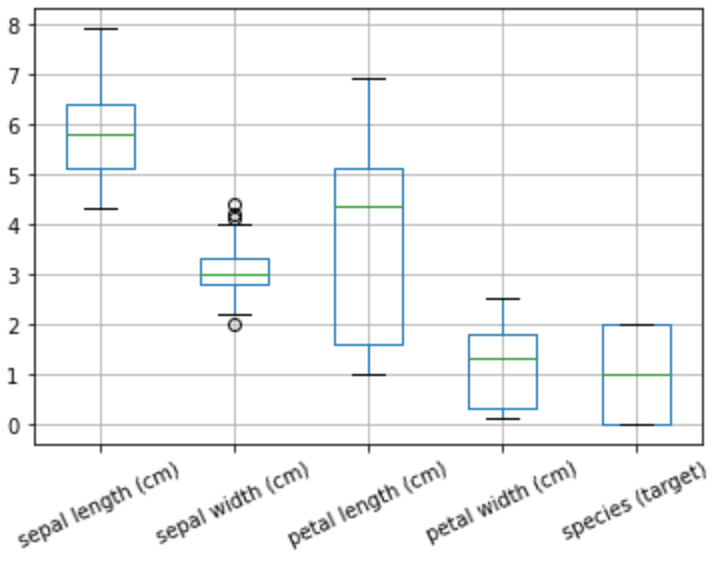
In [60]:
```
sample_data
```

Out[60]:
```
149    5.9
97     6.2
44     5.1
54     6.5
90     5.5
Name: sepal length (cm), dtype: float64
```

In [96]:
```
from sklearn.preprocessing import Normalizer
norm = Normalizer().fit(X)
X_norm = norm.transform(X)
norm_df = pd.DataFrame(X_norm)
norm_df.boxplot()
plt.show()
```



In [94]:
```
df.boxplot()
plt.xticks(rotation = 25)
plt.show()
```



Standardization

Like normalization, standardization is also required in some forms of machine learning when the input data points are scaled in different scales. Standardization can be a common scale for these data points.

We try to bring all the variables or features to a similar scale.Standardization rescales data(z) to have a mean (μ) of 0 and standard deviation (σ) of 1 (unit variance).It means centering the variable at zero.it is the process of rescaling the attributes so that they have **mean as 0 and variance as 1**.

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation } (x)}$$

```
from sklearn.preprocessing import StandardScaler
```

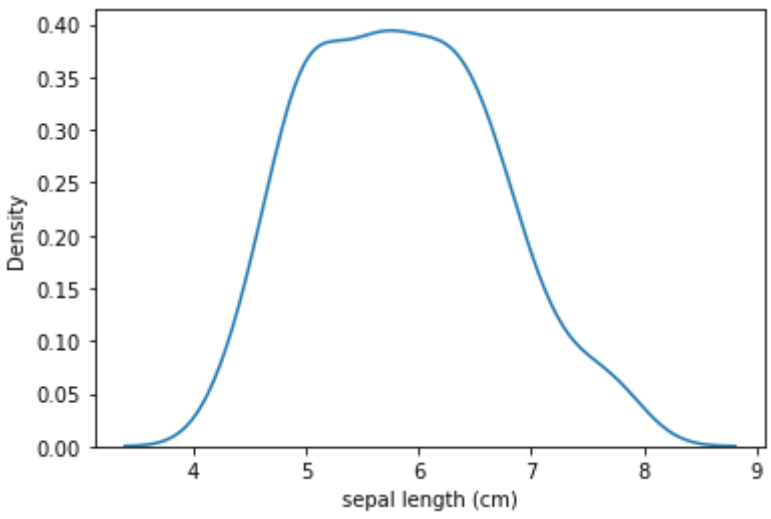
In [98]:
```
df.sample(5)
```

Out[98]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species (target)	species-names
146	6.3	2.5	5.0	1.9	2	virginica
62	6.0	2.2	4.0	1.0	1	versicolor
91	6.1	3.0	4.6	1.4	1	versicolor
82	5.8	2.7	3.9	1.2	1	versicolor
105	7.6	3.0	6.6	2.1	2	virginica

In [97]:
```
sns.kdeplot(df['sepal length (cm)'])
```

Out[97]:
```
<AxesSubplot:xlabel='sepal length (cm)', ylabel='Density'>
```



In [99]:
```
# Reading 5 values for standard deviation and mean
deviation = df.describe().loc[['std']][:4]
mean = df.describe().loc[['mean']][:4]
```

In [101…
```
mu = np.mean(X)
sigma = np.std(X)
mu, sigma
```

Out[101… `(3.4644999999999997, 1.9738430577598278)`

In [102…
```
sample_data = df['sepal length (cm)'].sample(5)
sample_data
```

Out[102…
```
47     4.6
91     6.1
78     6.0
110    6.5
20     5.4
Name: sepal length (cm), dtype: float64
```

In [104…
```
# After Scaling
ser = (df['sepal length (cm)'] - mu )/sigma
ser.loc[sample_data.index.values.tolist()]
```

Out[104…
```
47     0.575274
91     1.335213
78     1.284550
110    1.537863
20     0.980574
Name: sepal length (cm), dtype: float64
```

In [108…
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X)
X_transform = scaler.transform(X)

# fit in a dataframe
trans_df= pd.DataFrame(X_transform, columns=iris.feature_names)
trans_df.sample(4)
```

Out[108…

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
38	-1.748856	-0.131979	-1.397064	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
119	0.189830	-1.973554	0.705921	0.395774
95	-0.173674	-0.131979	0.251221	0.000878

In [109…
```
trans_df.describe().loc[['std','mean']].astype('int')
```

Out[109…

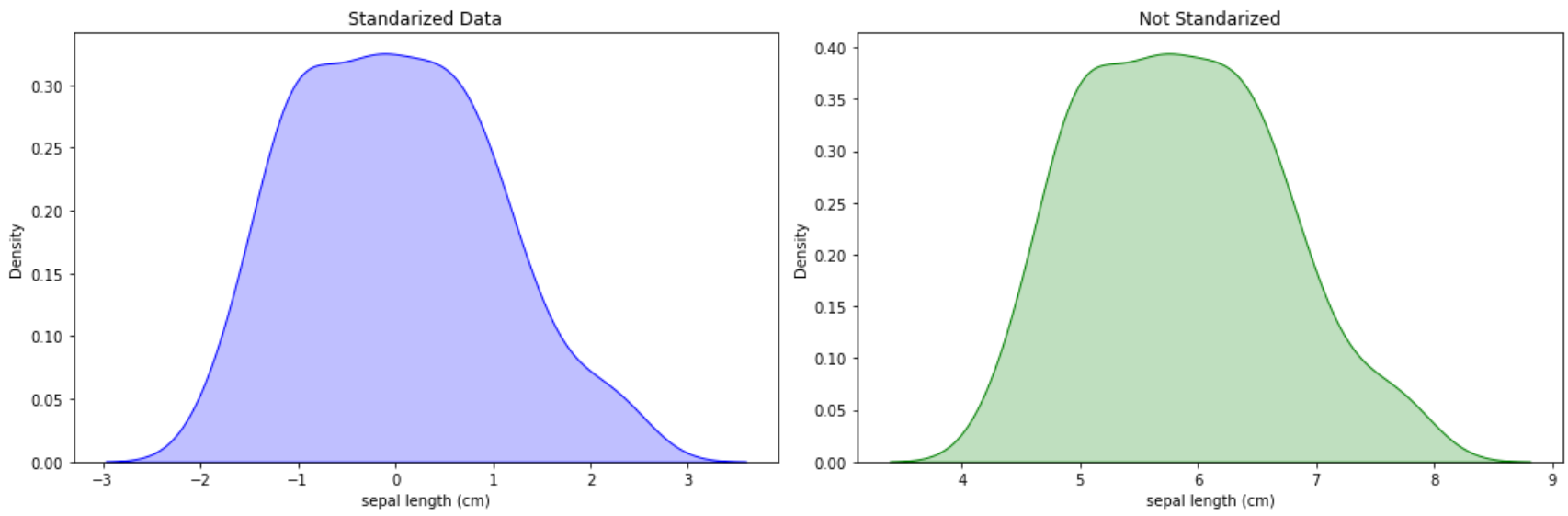
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
std	1	1	1	1
mean	0	0	0	0

In [110…
```
df.describe().loc[['std','mean']]
```

Out[110…

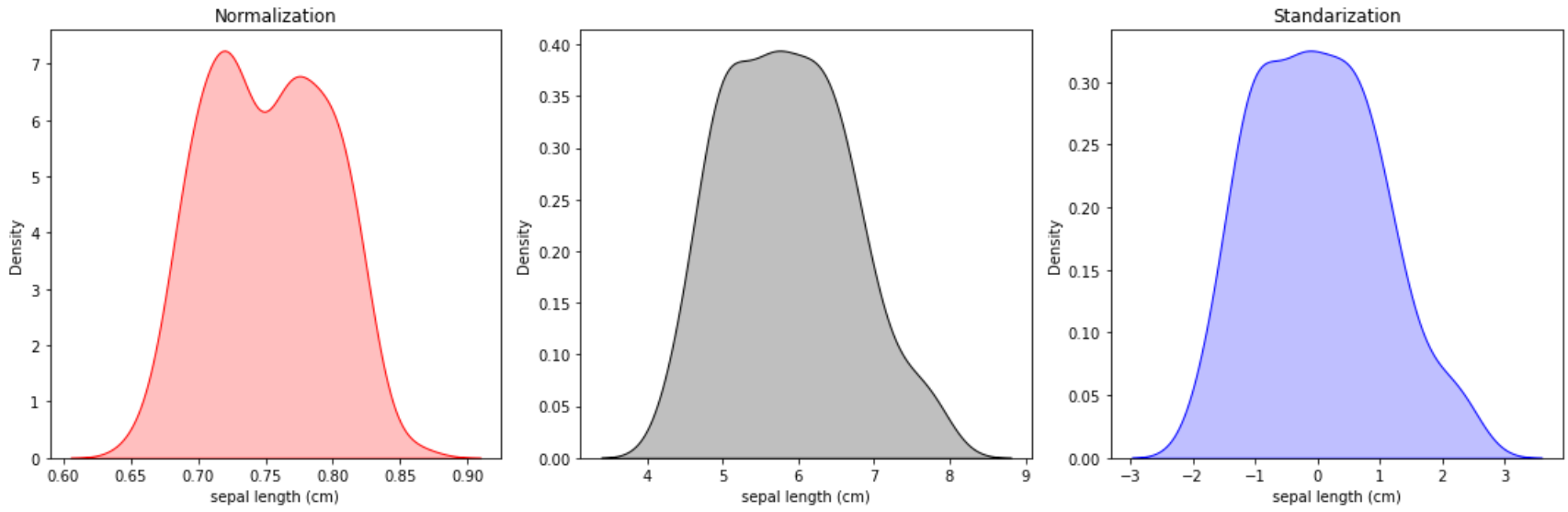
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species (target)
std	0.828066	0.435866	1.765298	0.762238	0.819232
mean	5.843333	3.057333	3.758000	1.199333	1.000000

In [114…
```
plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
plt.title('Standarized Data')
sns.kdeplot(trans_df['sepal length (cm)'], fill=True, color = 'blue')
plt.subplot(1, 2, 2)
plt.title('Not Standarized')
sns.kdeplot(df['sepal length (cm)'], fill=True, color = 'green')
plt.tight_layout()
```



In [124…

```
mini = np.min(X)
maxi = np.max(X)

sample_data = df['sepal length (cm)'].sample(5)

ser = (df['sepal length (cm)'] - mini )/(maxi - mini)
ser.loc[sample_data.index.values.tolist()]
from sklearn.preprocessing import Normalizer
norm = Normalizer().fit(X)
X_norm = norm.transform(X)
norm_df = pd.DataFrame(X_norm, columns=iris.feature_names)

plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.title('Normalization')
sns.kdeplot(norm_df['sepal length (cm)'], fill=True, color = 'red')
plt.subplot(1, 3, 2)
sns.kdeplot(df['sepal length (cm)'], fill=True, color = 'black')
plt.subplot(1, 3, 3)
plt.title('Standarization')
sns.kdeplot(trans_df['sepal length (cm)'], fill=True, color = 'blue')
plt.tight_layout()
```



Where to use Standardization ?

Since the results provided by the standardization are not bounded with any range as we have seen in normalization, it can be used with the data where the distribution is following the Gaussian distribution. In the case of outliers, standardization does not harm the position wherein normalization captures all the data points in their ranges.

Great Job!

That's all we need to know for now! Congratulations, you have learnt one more topic and hands-on with Python.This Notebook is prepared by Sumit Kumar Shukla IBM ICE.

About Author

Mr. Sumit is a Subject Matter Expert at IBM, and a data Scientist with more than five years of experience tutoring students from IITs, NITs, IISc, IIMs, and other prestigious institutions. Google Data Studio certified and IBM certified data analyst Data Science, Machine Learning Models, Graph Databases, and Data Mining techniques for Predictive Modeling and Analytics, as well as data integration, require expertise in Machine Learning and programming languages such as Python, R, and Tableau.

IBM