BITS Pilani

# Machine Learning (IS ZC464) Session 5 :

# Decision tree classifier and Data visualization for classification

# Discussion

- Decision tree classifier
- Data Visualization for classification
  - Example dataset
  - Data as tables-rows as instances and columns as features/ attributes
  - Scatter plots
  - Parallel coordinate graphs
  - Pearson correlation
  - Example code in Python

# Decision Tree

- A decision tree takes as input an object or situation described by a set of attributes and returns a decision.
- This decision is the predicted output value for the input.
- The input attributes  can be discrete or continuous.
- Classification Learning:
  - Learning a discrete valued function is called classification learning
- Regression :
  - Learning a continuous function is called Regression.

# Decision Tree

- A decision tree reaches its decision by performing a sequence of tests.
- All non leaf nodes lead to partial decisions and assist in moving towards the leaf node.
- Leaf nodes are the decisions based on properties satisfied at non leaf nodes on the path from the root node.

# Decision tree

- Leaf nodes depict the decision about a character having attributes falling on the path from the root node
- Each example that participate in the construction of the decision tree is called a training data and the complete set of the training data is called as training set.

# Limitations of Decision Tree Learning

- The tree memorizes the observations but does not extract any pattern from the examples.
- This limits the capability of the learning algorithm in that the observations do not extrapolate to examples it has not seen.

# How can we construct a decision tree for face recognition problem

- Define attributes
- Collect the attributes data from training samples
- Associate the output (to be used as leaf)

Imagine the size of decision tree with 1000 attributes capable of discriminating between persons!!!

# Decision trees

- The **attributes** aid in taking decisions.
- The most appropriate attribute is selected for testing in the beginning else the **size of the tree** becomes large resulting in large computational time.
- Leaf nodes represent the **decisions**.
- The attributes falling in the path from represent the  attributes fully able to define the decision at leaf.

# Goal Predicate: WillWait()

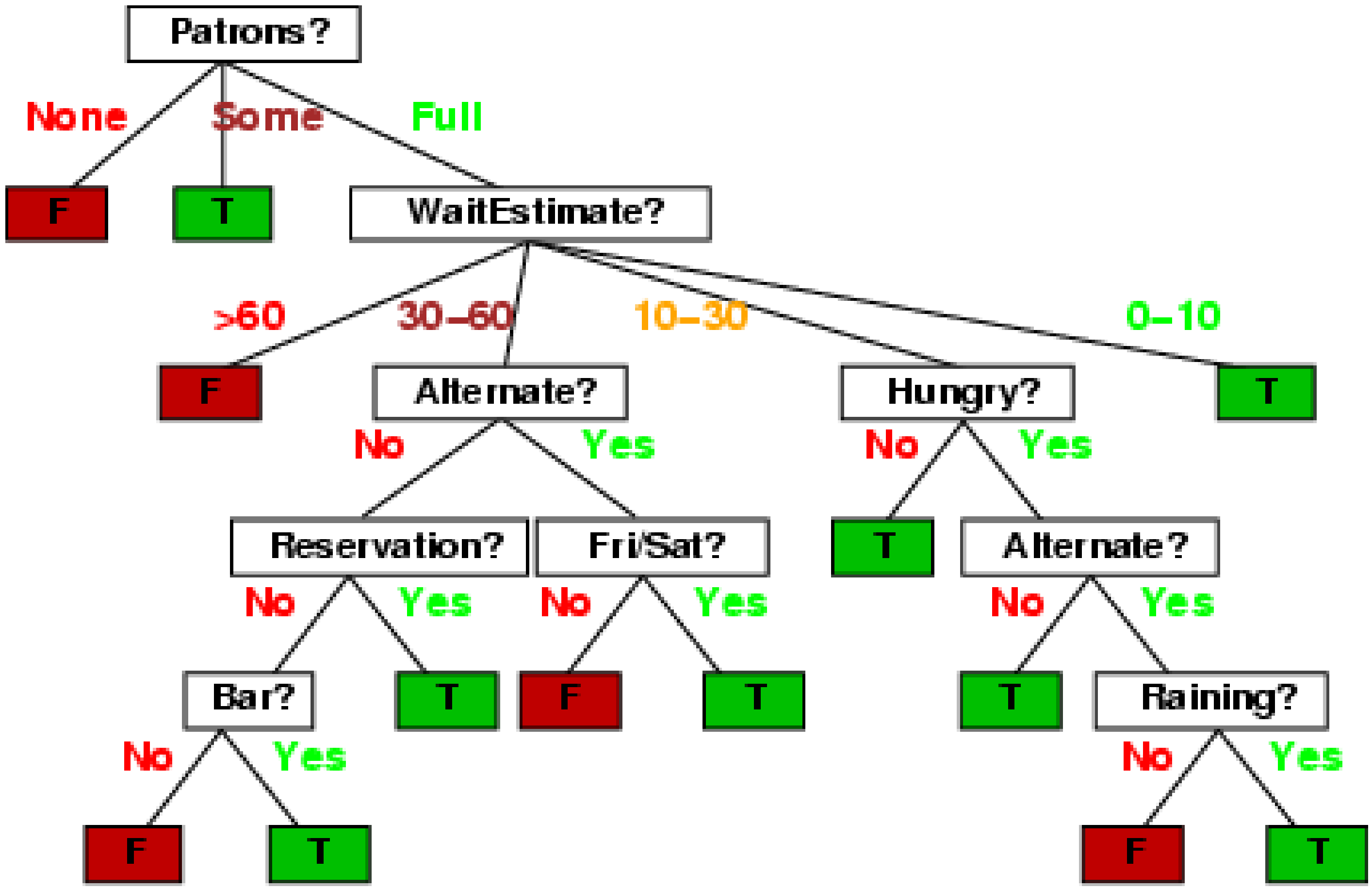
Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

# Attributes

| Example | Attributes | | | | | | | | | | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Alt* | *Bar* | *Fri* | *Hun* | *Pat* | *Price* | *Rain* | *Res* | *Type* | *Est* | *Wait* |
| $X_1$ | T | F | F | T | Some | $$$ | F | T | French | 0–10 | T |
| $X_2$ | T | F | F | T | Full | $ | F | F | Thai | 30–60 | F |
| $X_3$ | F | T | F | F | Some | $ | F | F | Burger | 0–10 | T |
| $X_4$ | T | F | T | T | Full | $ | F | F | Thai | 10–30 | T |
| $X_5$ | T | F | T | F | Full | $$$ | F | T | French | >60 | F |
| $X_6$ | F | T | F | T | Some | $$ | T | T | Italian | 0–10 | T |
| $X_7$ | F | T | F | F | None | $ | T | F | Burger | 0–10 | F |
| $X_8$ | F | F | F | T | Some | $$ | T | T | Thai | 0–10 | T |
| $X_9$ | F | T | T | F | Full | $ | T | F | Burger | >60 | F |
| $X_{10}$ | T | T | T | T | Full | $$$ | F | T | Italian | 10–30 | F |
| $X_{11}$ | F | F | F | F | None | $ | F | F | Thai | 0–10 | F |
| $X_{12}$ | T | T | T | T | Full | $ | F | F | Burger | 30–60 | T |

# Decision Tree




*Reference : aima.eecs.berkeley.edu/slides-**ppt**/m18-**learning.ppt***

# Size of the decision tree

- The **size of the Decision** tree depends on the **choice of the attributes** and the **order** in which they are used to test the examples.
- Selection  of attributes  must be "fairly good" and "really useless" attributes (such as type) should be avoided
- The **quality of the attribute** can be measured.
- One measure can be the **amount of information** the attribute carries.

# Information content

- If $v_i$ are different possible answers and $P(v_i)$are the probabilities that answer could be vi. Then the information content I of the actual answer is given by
  - $I(P(v_1), P(v_2), \ldots P(v_n)) = - \Sigma\, P(v_i)\log_2 P(v_i)$
- Assume that the training set contains 'p' positive examples and 'n' negative examples, then an estimate of the information contained in a correct answer is

$$I(p/(p+n), n/(p+n)) = - (p/(p+n))\log_2(p/(p+n)) - (n/(p+n))\log_2(n/(p+n))$$

# Refer the given table of Attributes

| Example | Attributes | | | | | | | | | | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Alt* | *Bar* | *Fri* | *Hun* | *Pat* | *Price* | *Rain* | *Res* | *Type* | *Est* | *Wait* |
| $X_1$ | T | F | F | T | Some | \$\$\$ | F | T | French | 0–10 | T |
| $X_2$ | T | F | F | T | Full | \$ | F | F | Thai | 30–60 | F |
| $X_3$ | F | T | F | F | Some | \$ | F | F | Burger | 0–10 | T |
| $X_4$ | T | F | T | T | Full | \$ | F | F | Thai | 10–30 | T |
| $X_5$ | T | F | T | F | Full | \$\$\$ | F | T | French | >60 | F |
| $X_6$ | F | T | F | T | Some | \$\$ | T | T | Italian | 0–10 | T |
| $X_7$ | F | T | F | F | None | \$ | T | F | Burger | 0–10 | F |
| $X_8$ | F | F | F | T | Some | \$\$ | T | T | Thai | 0–10 | T |
| $X_9$ | F | T | T | F | Full | \$ | T | F | Burger | >60 | F |
| $X_{10}$ | T | T | T | T | Full | \$\$\$ | F | T | Italian | 10–30 | F |
| $X_{11}$ | F | F | F | F | None | \$ | F | F | Thai | 0–10 | F |
| $X_{12}$ | T | T | T | T | Full | \$ | F | F | Burger | 30–60 | T |

# Information content

- Since

  $I(p/(p+n), n/(p+n)) = -(p/(p+n))\log_2(p/(p+n))$
  $\qquad - (n/(p+n))\log_2(n/(p+n))$

  - information $= -(6/12)\log_2(1/2) - (6/12)\log_2(1/2)$
  - $= -\log_2(1/2)$

    $= \log_2(1/2)^{-1}$

    $= \log_2(2)$

    $= 1$ bit

# Generalize the splitting

- Let the attribute A divides the entire training set into sets E1, E2, … Ev. Where v is the total number of values A can be tested on.
- Assume that each set $E_i$ contains $p_i$ positive examples and $n_i$ negative examples
- **Remainder (A)**

  **$= \Sigma\ (p_i+n_i)/(p+n)\ I(p_i/(p_i+n_i),\ n_i/(p_i+n_i))$**

  **over i=1 to v**

# Gain(A)

- Gain(A)

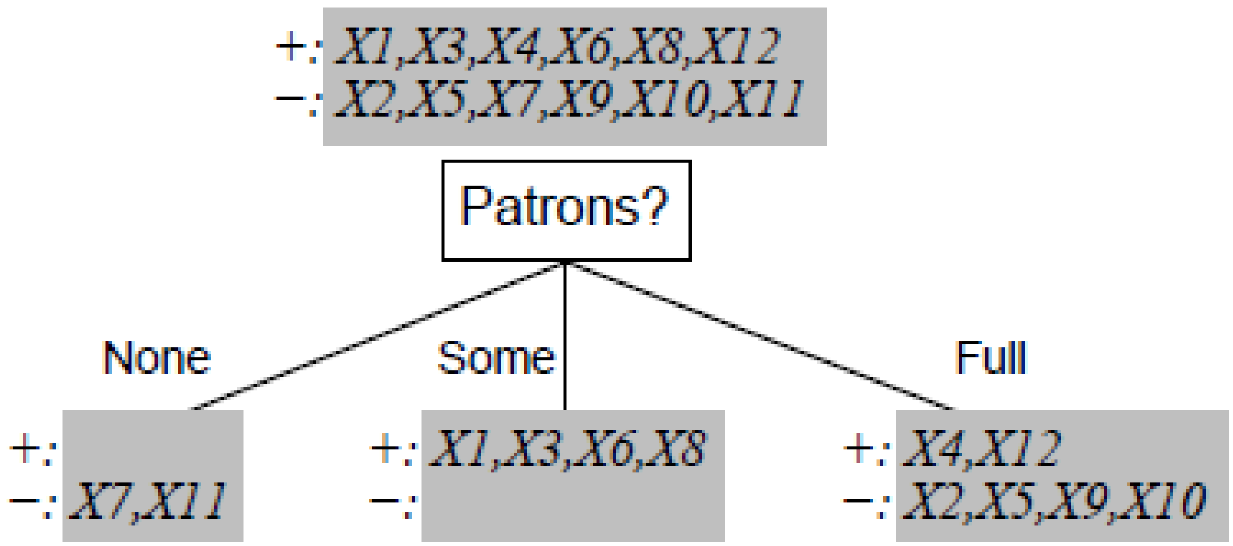
  $= I(p/p+n, n/p+n) – \text{Remainder}(A)$

The heuristic to choose attribute A from a set of all attributes is the maximum gain

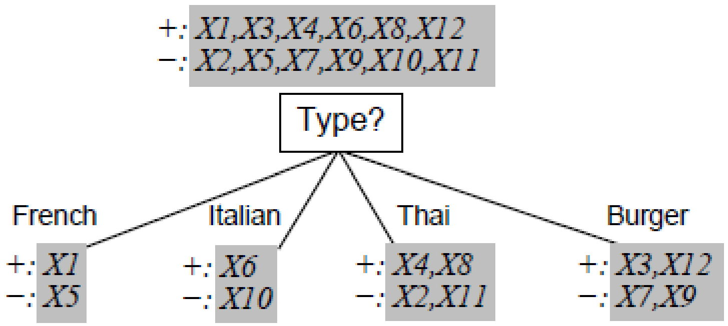
**Compute**

1. **Gain(Patrons)**
2. **Gain(type)**

# Selecting patrons attribute



# Selecting type as attribute



# Gain(patron)

- 1 – ((2/12)I(o,1) + (4/12)I(1, 0) + (6/12) I(2/6, 4/6))
- Approximately equal to 0.541 bits

# Refer the given table of Attributes and compute Gain

| Example | Attributes | | | | | | | | | | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Alt* | *Bar* | *Fri* | *Hun* | *Pat* | *Price* | *Rain* | *Res* | *Type* | *Est* | *Wait* |
| $X_1$ | T | F | F | T | Some | $$$ | F | T | French | 0–10 | T |
| $X_2$ | T | F | F | T | Full | $ | F | F | Thai | 30–60 | F |
| $X_3$ | F | T | F | F | Some | $ | F | F | Burger | 0–10 | T |
| $X_4$ | T | F | T | T | Full | $ | F | F | Thai | 10–30 | T |
| $X_5$ | T | F | T | F | Full | $$$ | F | T | French | >60 | F |
| $X_6$ | F | T | F | T | Some | $$ | T | T | Italian | 0–10 | T |
| $X_7$ | F | T | F | F | None | $ | T | F | Burger | 0–10 | F |
| $X_8$ | F | F | F | T | Some | $$ | T | T | Thai | 0–10 | T |
| $X_9$ | F | T | T | F | Full | $ | T | F | Burger | >60 | F |
| $X_{10}$ | T | T | T | T | Full | $$$ | F | T | Italian | 10–30 | F |
| $X_{11}$ | F | F | F | F | None | $ | F | F | Thai | 0–10 | F |
| $X_{12}$ | T | T | T | T | Full | $ | F | F | Burger | 30–60 | T |

# Decision Trees

- Learning is through a series of decisions taken with respect to the attribute at the non-leaf node.
- There can be **many trees** possible for the given training data.
- Finding the smallest DT is an NP-complete problem.
- Greedy selection of the attribute  with largest gain to split the training data into two or more sub-classes may lead to approximately the smallest tree

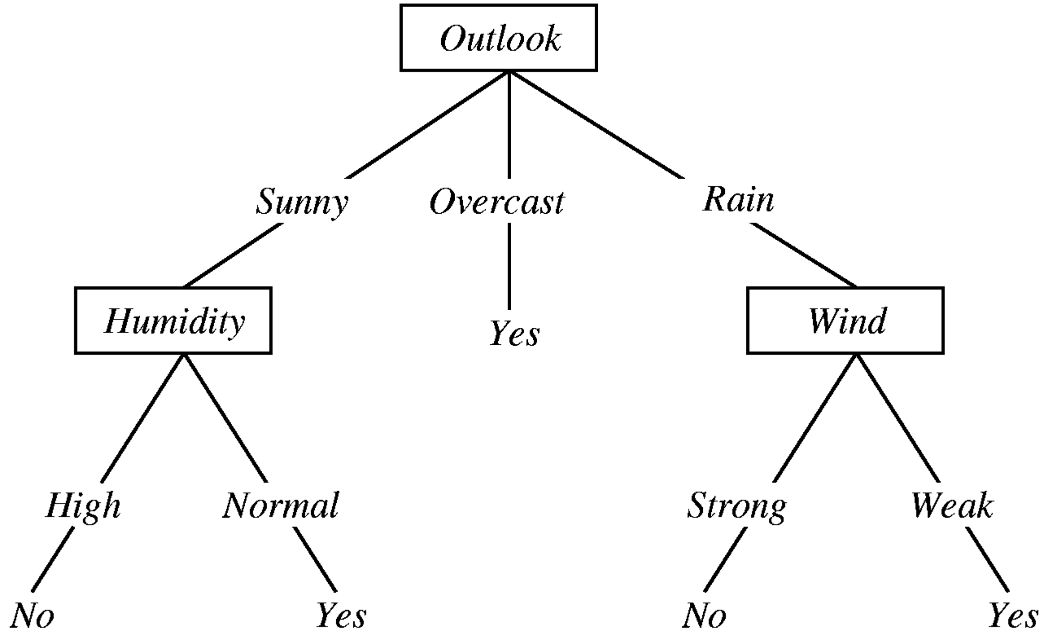
# Decision Trees

- If the decisions are binary, then in the best case the decision eliminates almost half of the regions (leaves).
- If there are ‘b’ regions, then the correct region can be found in $\log_2(b)$ decisions in the best case.
- The height of the decision trees depends on the order of the attributes selected to split the training examples at each step.

# Expressiveness of the DS

- A decision tree can represent a disjunction of conjunctions of constraints on the attribute values of instances.
  - Each path corresponds to a conjunction
  - The tree itself corresponds to a disjunction

# Example




If (O=Sunny AND H=Normal) OR (O=Overcast) OR (O=Rain AND W=Weak) then YES

- “A disjunction of conjunctions of constraints on attribute values”

# Entropy

- It is the measure of the information content and is given by
  - $I = - \Sigma P(v_i)\log_2 P(v_i)$
  - Where v1,v2,..,vk are the values of the attribute on which the decisions bifurcate.

| rec | Age | Income | Student | Credit_rating | Buys_computer |
|---|---|---|---|---|---|
| r1 | <=30 | High | No | Fair | No |
| r2 | <=30 | High | No | Excellent | No |
| r3 | 31…40 | High | No | Fair | Yes |
| r4 | >40 | Medium | No | Fair | Yes |
| r5 | >40 | Low | Yes | Fair | Yes |
| r6 | >40 | Low | Yes | Excellent | No |
| r7 | 31…40 | Low | Yes | Excellent | Yes |
| r8 | <=30 | Medium | No | Fair | No |
| r9 | <=30 | Low | Yes | Fair | Yes |
| r10 | >40 | Medium | Yes | Fair | Yes |
| r11 | <.=30 | Medium | Yes | Excellent | Yes |
| r12 | 31…40 | Medium | No | Excellent | Yes |
| r13 | 31…40 | High | Yes | Fair | Yes |
| r14 | >40 | Medium | No | Excellent | No |

# Class Work

**Remainder (A)**
**$= \Sigma (p_i+n_i)/(p+n)$**
**$I(p_i/(p_i+n_i), n_i/(p_i+n_i))$**
**over i=1 to v**

- Identify the examples belonging to the two sets constructed after the data is split on the basis of **attribute 'student'.**
- Compute the **total information content** of the training data.
- Compute the **information gain** if the training data is split on the basis of the attribute 'student'.
- Draw the **decision tree**, which may or may not be optimal.

# Understand the examples

- Decisions are binary – yes / no
- Training data as <example, decision> pair
- <r1,no>, <r2,no>, <r3,yes>, <r4,yes> and so on
- Positive examples: r3, r4, r5, r7, r9, r10, r11, r12, r13
- Negative examples: r1,r2,r6, r8, r14
- Is the given training set sufficient to take any decision?
- Is the generalization capability of the given training set sufficient?

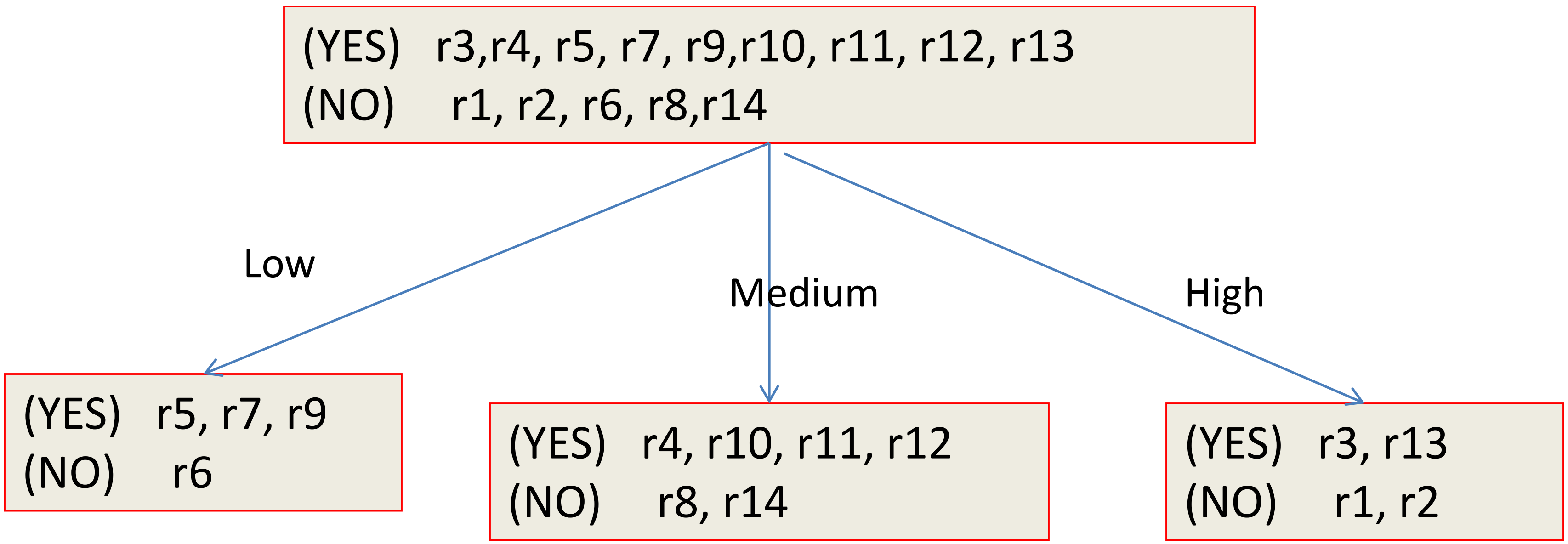
# Information content of the given training data

- Here v1 = yes, v2 = no
- Positive examples: r3, r4, r5, r7, r9, r10, r11, r12, r13
- Negative examples: r1,r2,r6, r8, r14
- Total number of examaples = 14
- P(v1) = 9/14,  P(v2)=5/14
- Information content is represented by the notion I(9/14, 5/14)
- Entropy = $- (P(v_1)\log_2(P(v_1)) + P(v_2)\log_2(P(v_2)) )$

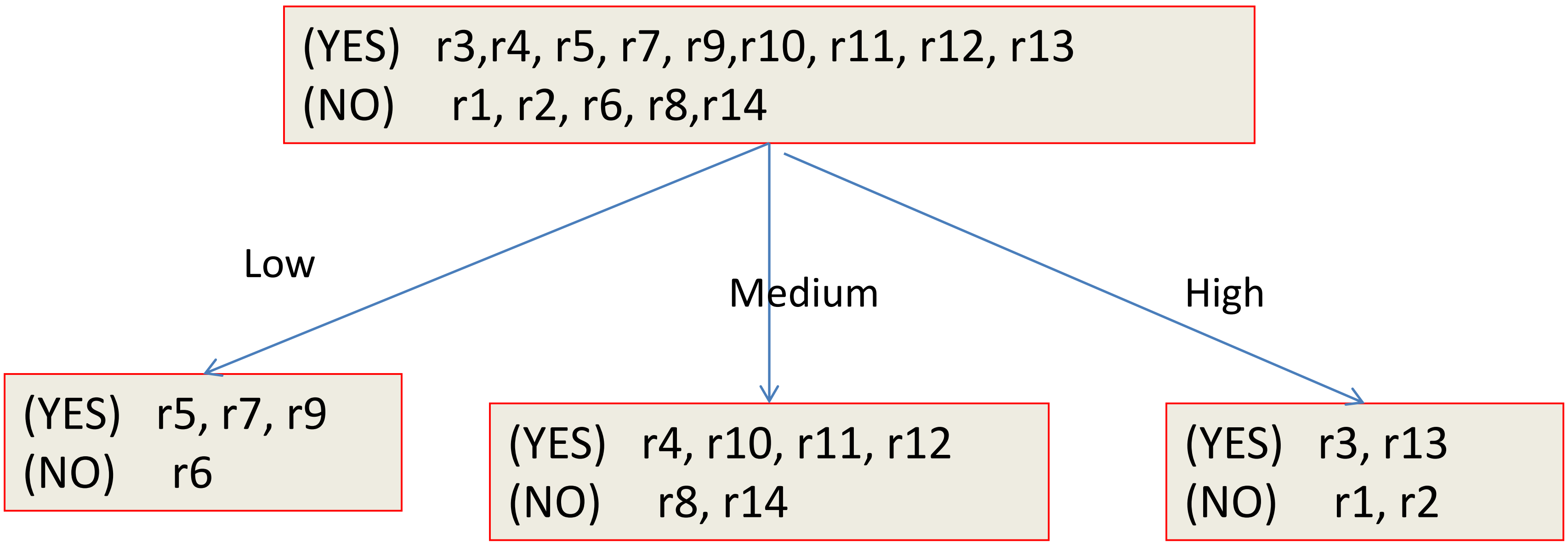
  $= -((9/14)* \log_2 (9/14) + (5/14)*\log_2 (5/14))$

  = 0.8108

# Compute the significance of attribute ‘income’



# Compute the significance of attribute ‘income’




Observe that the split regions of examples possess mixed decisions, this shows the poor quality of the attribute ‘income’

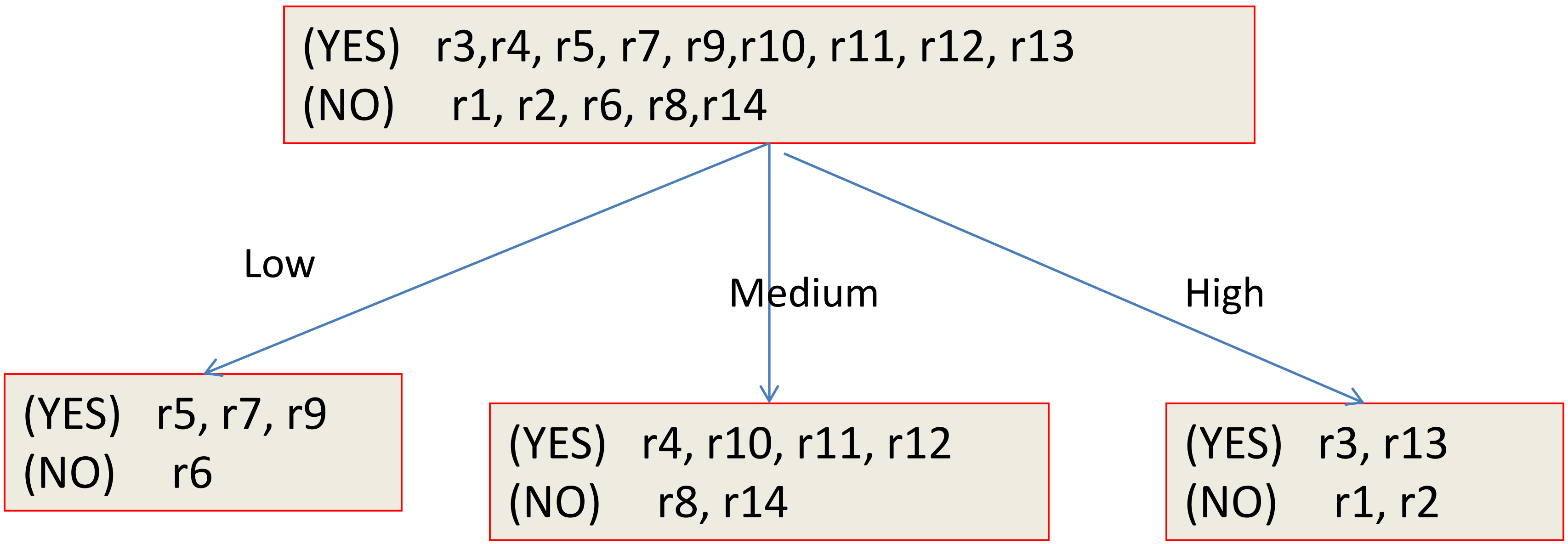
# Recall Generalize the splitting

- Let the attribute A divides the entire training set into sets E1, E2, … Ev. Where v is the total number of values  A can be tested on.
- Assume that each set $E_i$ contains $p_i$ positive examples and $n_i$ negative examples
- **Remainder (A)**

  **$= \Sigma\ (p_i+n_i)/(p+n)\ I(p_i/(p_i+n_i), n_i/(p_i+n_i))$**

  **over i=1 to v**

# Compute the significance of attribute ‘income’




**$p_1 = 3$**
**$n_1 = 1$**

**$p_2 = 4$**
**$n_2 = 2$**

**$p_3 = 2$**
**$n_3 = 2$**

Compute attribute

**Remainder (A)**

**$= \Sigma (p_i+n_i)/(p+n) \, I(p_i/(p_i+n_i), n_i/(p_i+n_i))$ over i=1 to v**

- Remainder = (4/14)*I(3/4,1/4)

  + (6/14)*I(4/6, 2/6)

  +(4/14)*I(2/4,2/4)

$= (4/14)\ \{-(3/4) \log_2(3/4) - (1/4)\log_2 (1/4)\}$

$+(6/14)\ \{-(4/6) \log_2(4/6) - (2/6)\log_2 (2/6)\}$

$+(4/14)\{ (-(2/4)\log_2 (2/4)-(2/4)\log_2(2/4)$

[Home Work: Remaining computation]

**$p_1 = 3$**
**$n_1 = 1$**

**$p_2 = 4$**
**$n_2 = 2$**

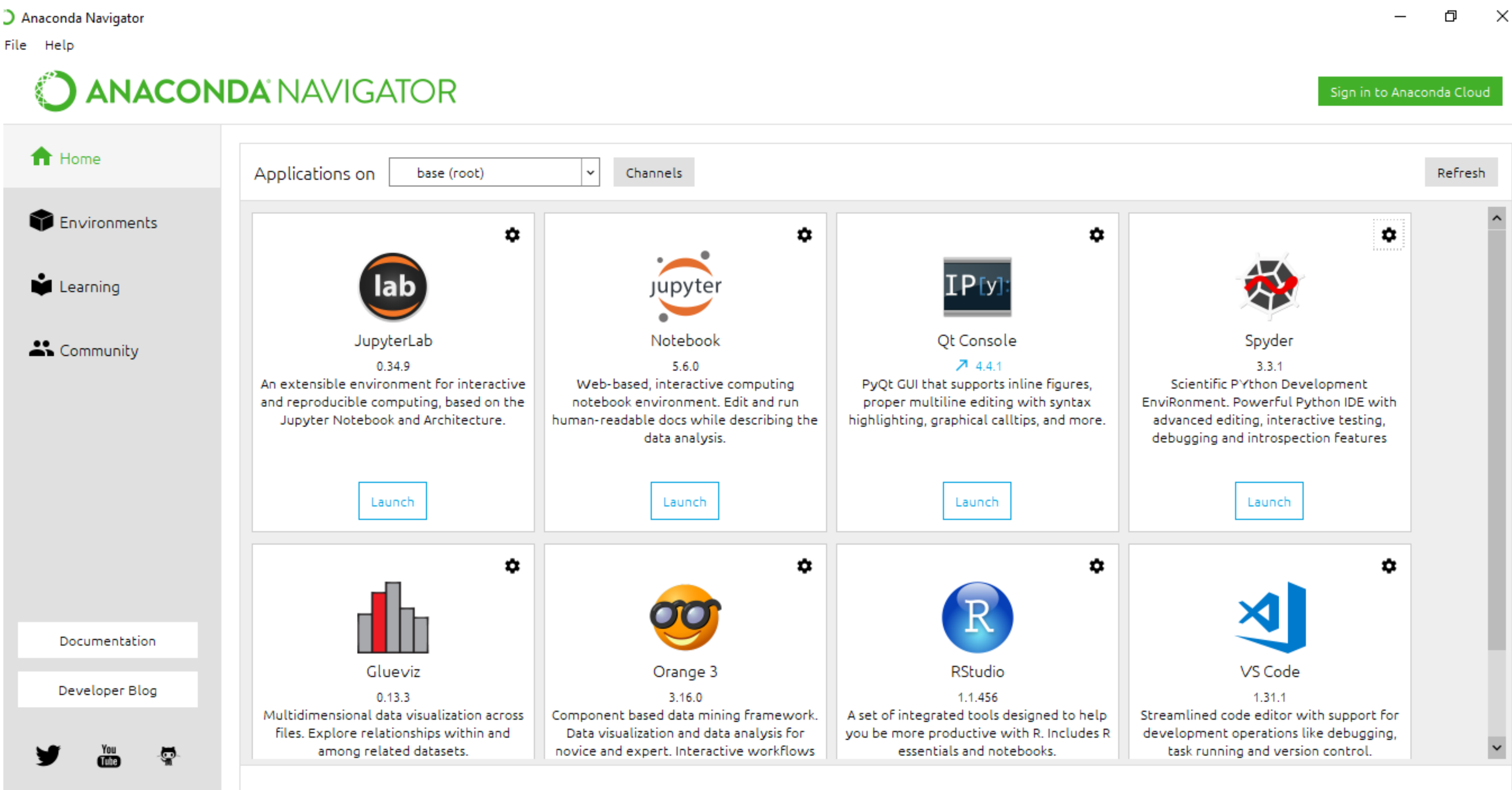
**$p_3 = 2$**
**$n_3 = 2$**

# Using Python to understand data

- Python Standard library: https://www.python.org/
- Scientific Computing: http://www.numpy.org/
- Machine learning software: https://scikit-learn.org/
- Data analysis library: http://pandas.pydata.org/
- Plotting library: https://matplotlib.org/

- Complete collection available through anaconda software available free at https://www.anaconda.com/

# Anaconda Navigator



# Example datasets

- UCI machine learning repository

https://archive.ics.uci.edu/ml/datasets.html

- Used frequently by beginners in machine learning: *Iris data set*

https://archive.ics.uci.edu/ml/datasets/iris

# IRIS dataset

- Input attributes : Sepal length, sepal width, petal length and petal width
- Class labels: Iris Setosa, Iris Versicolour and Iris Virginica


Iris Versicolor Iris Setosa Iris Virginica

# Data samples for Setosa class

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|---|---|---|---|
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |

# Data samples for virginica class

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|---|---|---|---|
| 6.3 | 3.3 | 6.0 | 2.5 | virginica |
| 5.8 | 2.7 | 5.1 | 1.9 | virginica |
| 7.1 | 3.0 | 5.9 | 2.1 | virginica |
| 6.3 | 2.9 | 5.6 | 1.8 | virginica |
| 6.5 | 3.0 | 5.8 | 2.2 | virginica |

# Data samples for versicolor class

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|---|---|---|---|
| 7.0 | 3.2 | 4.7 | 1.4 | versicolor |
| 6.4 | 3.2 | 4.5 | 1.5 | versicolor |
| 6.9 | 3.1 | 4.9 | 1.5 | versicolor |
| 5.5 | 2.3 | 4.0 | 1.3 | versicolor |
| 6.5 | 2.8 | 4.6 | 1.5 | versicolor |

# Iris Data

- Number of samples per class = 50
- Total number of records = 150 (for 3 classes)

# Working with the data : code and output

```python
#using pandas for iris data
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
dataset_url = ("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data")
datarv= pd.read_csv(dataset_url, header=None, prefix="V")
print(datarv.head())
print(datarv.tail())
summary = datarv.describe()
print(summary)
```

```
    V0   V1   V2   V3           V4
0  5.1  3.5  1.4  0.2  Iris-setosa
1  4.9  3.0  1.4  0.2  Iris-setosa
2  4.7  3.2  1.3  0.2  Iris-setosa
3  4.6  3.1  1.5  0.2  Iris-setosa
4  5.0  3.6  1.4  0.2  Iris-setosa
      V0   V1   V2   V3              V4
145  6.7  3.0  5.2  2.3  Iris-virginica
146  6.3  2.5  5.0  1.9  Iris-virginica
147  6.5  3.0  5.2  2.0  Iris-virginica
148  6.2  3.4  5.4  2.3  Iris-virginica
149  5.9  3.0  5.1  1.8  Iris-virginica
               V0          V1          V2          V3
count  150.000000  150.000000  150.000000  150.000000
mean     5.843333    3.054000    3.758667    1.198667
std      0.828066    0.433594    1.764420    0.763161
min      4.300000    2.000000    1.000000    0.100000
25%      5.100000    2.800000    1.600000    0.300000
50%      5.800000    3.000000    4.350000    1.300000
75%      6.400000    3.300000    5.100000    1.800000
max      7.900000    4.400000    6.900000    2.500000
```

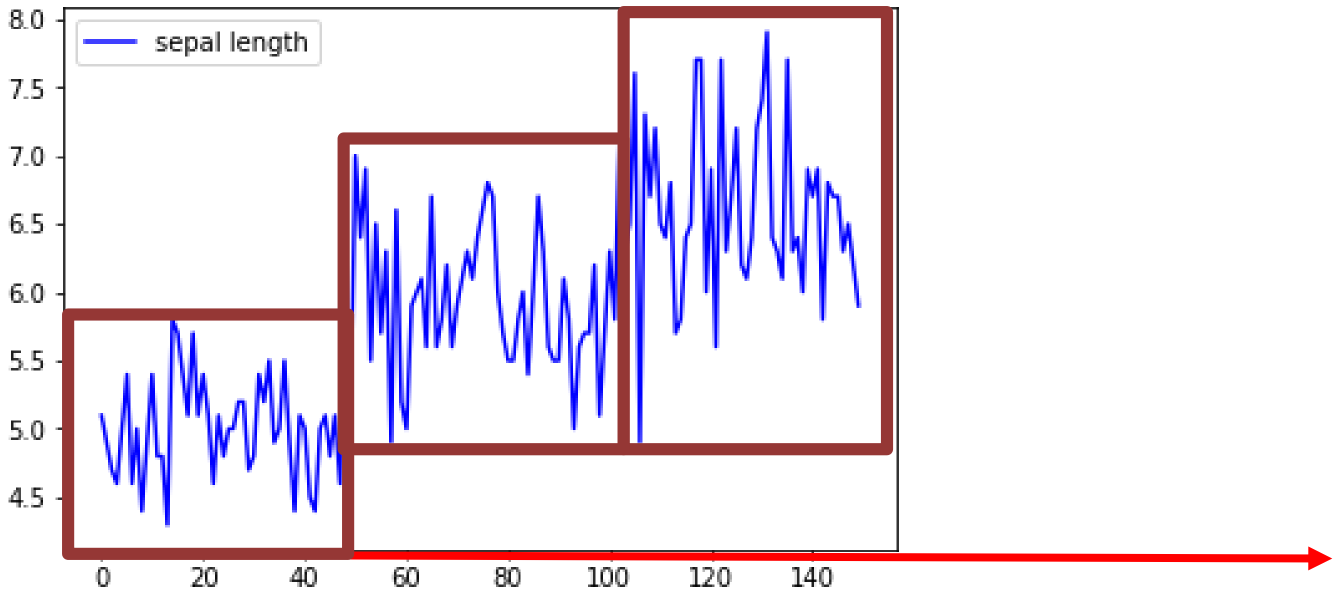
V0, V1, V2, V3: attributes
V4-class label

Column numbers

Summary of the data

# Visualizing the attribute's ability to discriminate the classes

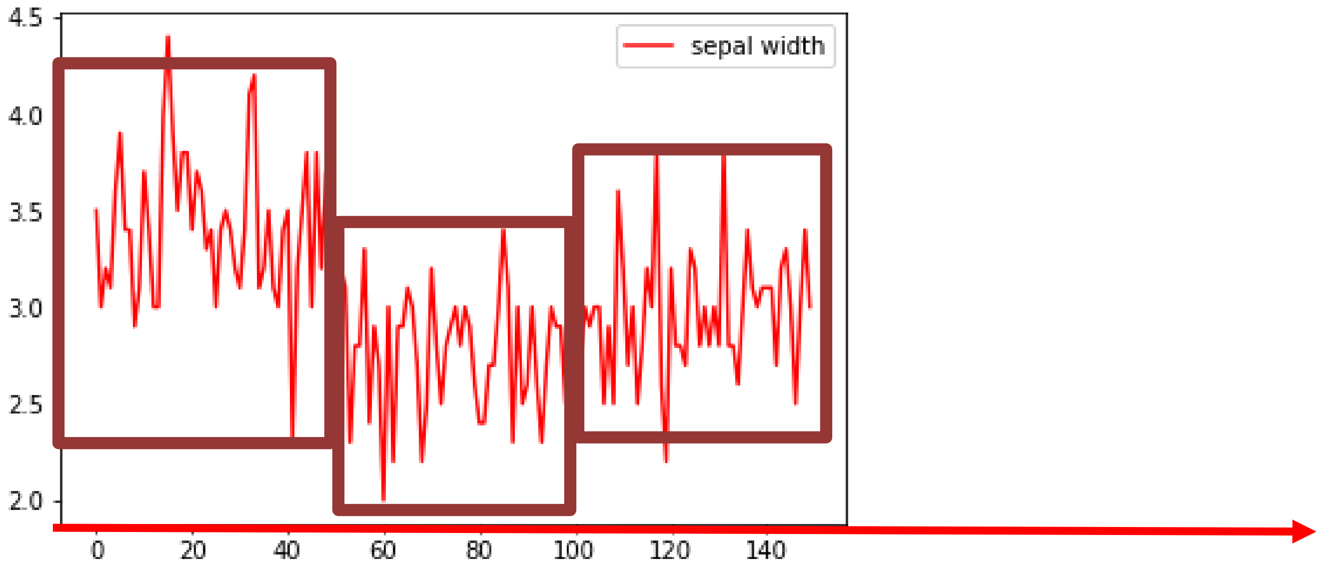
```
datarow = datarv.iloc[:,0]
datarow.plot(color ="blue", label = 'sepal length')
plot.legend()
```




Sample number

# Visualizing the attribute's ability to discriminate the classes

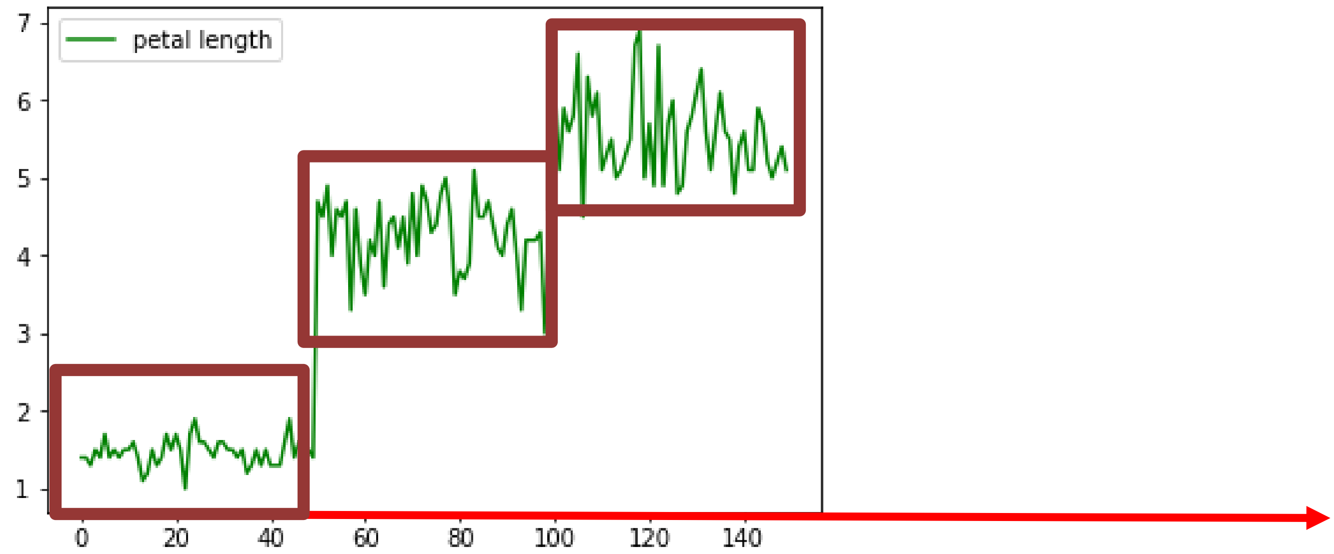
```
datarow = datarv.iloc[:,1]
datarow.plot(color ="red", label = 'sepal width')
plot.legend()
```




Sample number

# Visualizing the attribute's ability to discriminate the classes

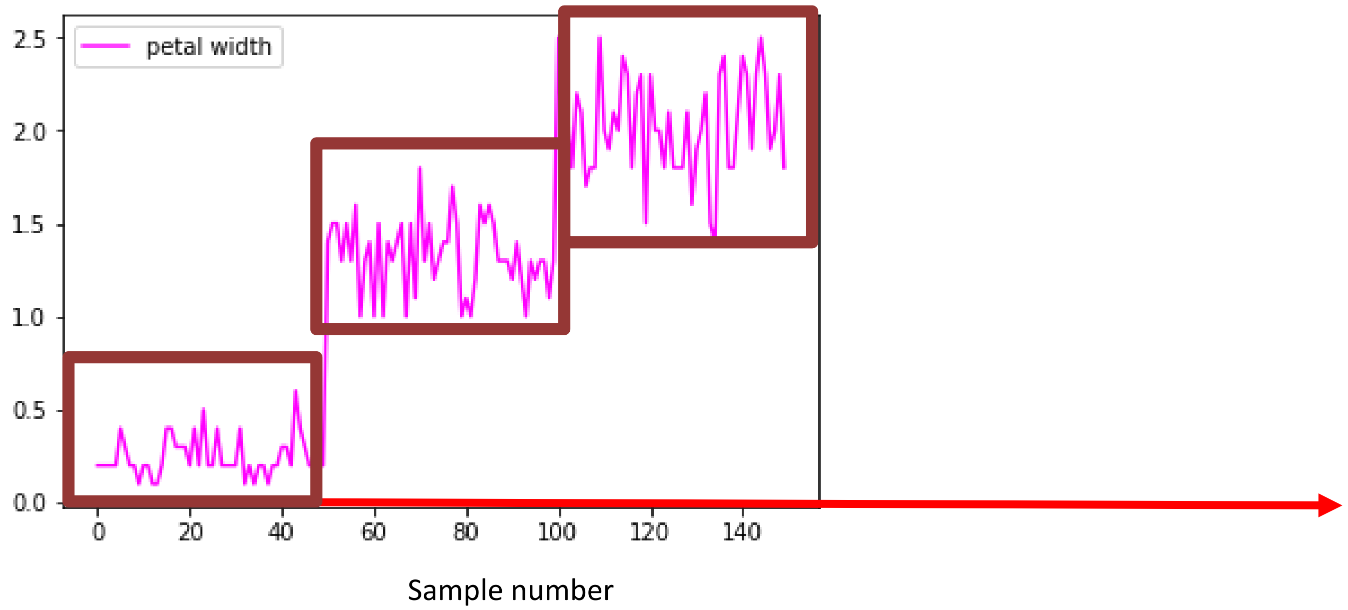
```
datarow = datarv.iloc[:,2]
datarow.plot(color ="green", label = 'petal length')
plot.legend()
```




Sample number

# Visualizing the attribute's ability to discriminate the classes

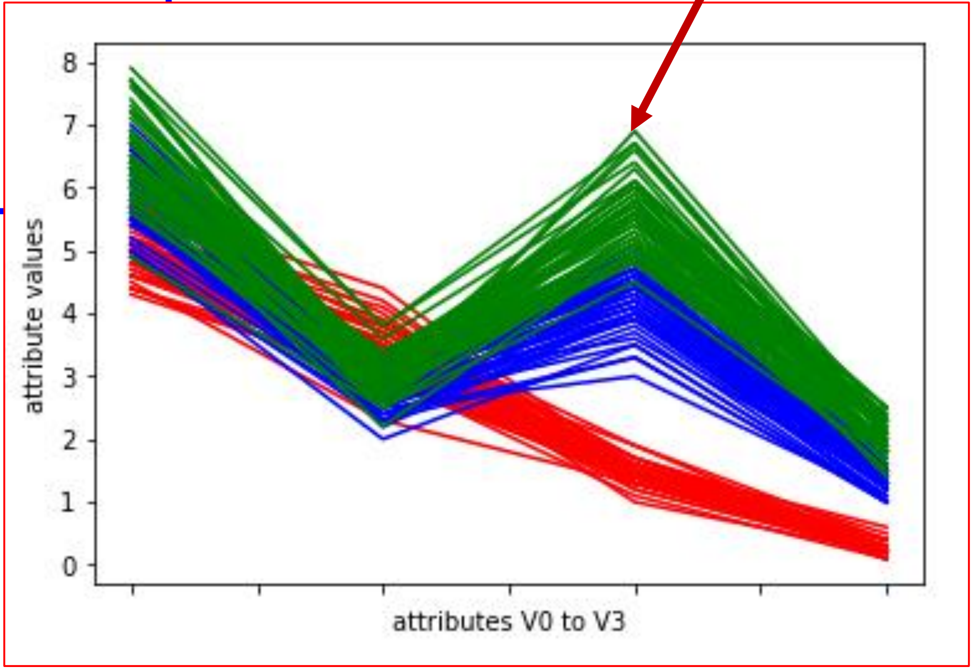
```
datarow = datarv.iloc[:,3]
datarow.plot(color ="magenta", label = 'petal width')
plot.legend()
```



# Parallel Coordinate graph

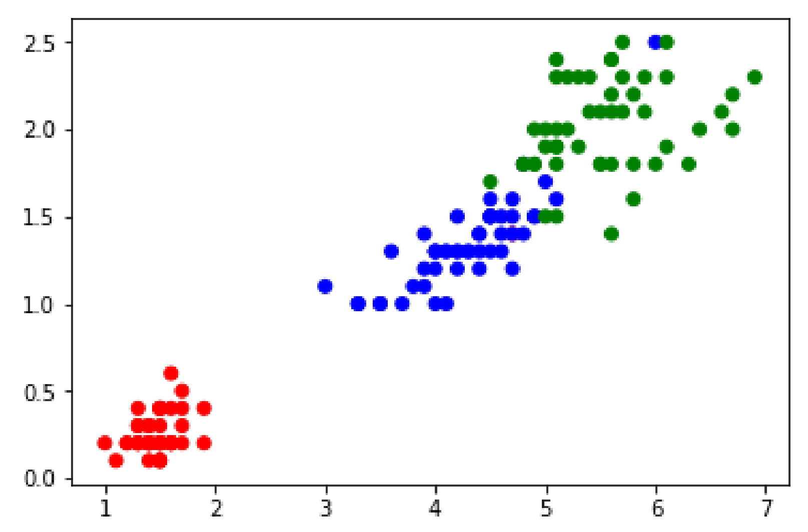
```
for i in range(150):
    if datarv.iat[i,4]=="Iris-setosa":
        pcolor = "red"
    elif datarv.iat[i,4]=="Iris-virginica":
        pcolor ="green"
    else:
        pcolor = "blue"
    datarow = datarv.iloc[i,0:4]
    datarow.plot(color = pcolor)

plot.xlabel("attributes V0 to V3")
plot.ylabel("attribute values")
```

Most discriminative feature (A2), i.e. petal length



# Scatter Plot

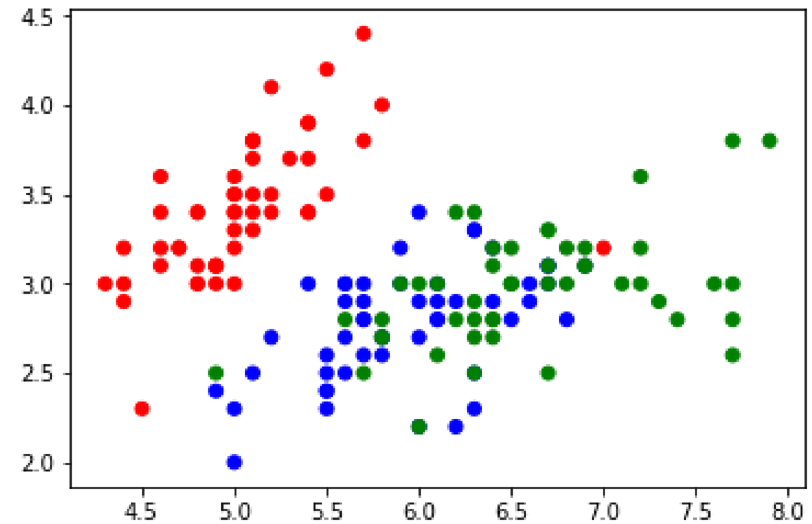
```
colorarr = []
for i in range(150):
    if datarv.iat[i,4]=="Iris-setosa":
        pcolor = "red"
    elif datarv.iat[i,4]=="Iris-virginica":
        pcolor ="green"
    else:
        pcolor = "blue"
    colorarr = np.append(colorarr, pcolor)

datacol0 = datarv.iloc[1:150,2]
datacol1 = datarv.iloc[1:150,3]
plot.scatter(datacol0,datacol1,c = colorarr)
```



Import numpy as np

# Scatter Plot

```
colorarr = []
for i in range(150):
    if datarv.iat[i,4]=="Iris-setosa":
        pcolor = "red"
    elif datarv.iat[i,4]=="Iris-virginica":
        pcolor ="green"
    else:
        pcolor = "blue"
    colorarr = np.append(colorarr, pcolor)

datacol0 = datarv.iloc[1:150,0]
datacol1 = datarv.iloc[1:150,1]
plot.scatter(datacol0,datacol1,c = colorarr)
```



# Using IRIS data for classification

```python
#using pandas for iris data
import pandas as pd
from pandas import DataFrame
import numpy as np
import random
import matplotlib.pyplot as plot
from sklearn.linear_model import LogisticRegression as Model
dataset_url = ("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data")
datarv= pd.read_csv(dataset_url, header=None, prefix="V")
```

```python
#using various print statements for understanding the data
#----------------------------------
#print(datarv.head())
#print(datarv.tail())
summary = datarv.describe()
#print(summary)
#----------------------------------
```

```python
def train(features, target):
    classifier_model = Model()
    classifier_model.fit(features, target)
    return classifier_model
```

```python

def predict(model, testfeatures):
    pred = model.predict(testfeatures)
    return pred

def evaluate_acc(pred, target):
    a = pred.shape
#    b = target.shape

#    print(a)
#    print(b)
#    print(a[0])
    acc_count = 0;
    for i in range(0,a[0]):
        if(pred[i]==target.iloc[i]):
            acc_count = acc_count+1
    accuracy_percent = (acc_count/a[0])*100
    return accuracy_percent
```

# Contd...

```python

N = 150
sum_crossfold = 0
for i in range(10):
    N_train = int(np.floor(0.7*N))
#     print(N_train)
    idx = np.random.permutation(N)
#     print(idx)
    # Below splitting the data into training and test indices
    idx_train = idx[:N_train]
    idx_test = idx[N_train:]
#     print(idx_train)
#     print(idx_test)
    # check the dimension of your data
    #IRIS data has 150 rows and 5 colums of which first four are the attributes
    # and the 5th column has class label
    #print(datarv.shape)
    # Use the first four columns as features
    features_train = datarv.iloc[idx_train,0:3]
    #print(features_train)
    features_test = datarv.iloc[idx_test, 0:3]
    #print(features_test)
    # extract the targets similarly
    target_train = datarv.iloc[idx_train,4]
    #print(target_train)
    target_test = datarv.iloc[idx_test, 4]
    #print(target_test)
    model = train(features_train, target_train)
    predicted_classes = predict(model, features_test)
#     print(predicted_classes)
    accuracy = evaluate_acc(predicted_classes, target_test)
    sum_crossfold =sum_crossfold + accuracy
    print(i,accuracy)

print(sum_crossfold/10)
```

Using all 4 features

output

```
0 88.88888888888889
1 91.11111111111111
2 97.77777777777777
3 95.55555555555556
4 93.33333333333333
5 93.33333333333333
6 95.55555555555556
7 91.11111111111111
8 91.11111111111111
9 86.66666666666667
92.44444444444443
```

# Pearson Correlation coefficient

- The degree of correlation between two attributes or one attribute and the target can be quantified using Pearson's correlation coefficient.
- Let u and v be the two equal length vectors

$u = \langle u_1, u_2, u_3, \ldots, u_n \rangle$

$v = \langle v_1, v_2, v_3, \ldots, v_n \rangle$

- Let $M_u$ and $M_v$ be the mean values of vectors u and v respectively

# Pearson Correlation coefficient

- Compute the vector by taking the difference of each value with their corresponding mean as follows

$\Delta u = < u_1 - M_u , u_2 - M_u , u_3 - M_u , \ldots, u_n - M_u >$

$\Delta v = < v_1 - M_v , v_2 - M_v , v_3 - M_v , \ldots, v_n - M_v >$

- The Correlation between u and v is defined as below

$$Corr(u, v) = \frac{\Delta u^T \Delta v}{\sqrt{(\Delta u^T \Delta u) * (\Delta v^T \Delta v)}}$$

# Python Code for Pearson Correlation Coefficient

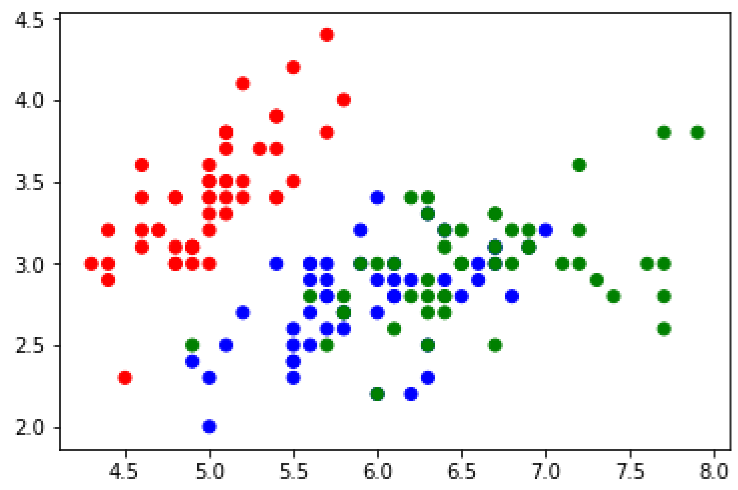
```python
#using pandas for iris data
import pandas as pd
from pandas import DataFrame
import numpy as np
import random
import matplotlib.pyplot as plot
dataset_url = ("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data")
datarv= pd.read_csv(dataset_url, header=None, prefix="V")

featureVec0 = datarv.iloc[:,0]
featureVec1 = datarv.iloc[:,1]
featureVec2 = datarv.iloc[:,2]
featureVec3 = datarv.iloc[:,3]

colorarr = []
for i in range(150):
    if datarv.iat[i,4]=="Iris-setosa":
        pcolor = "red"
    elif datarv.iat[i,4]=="Iris-virginica":
        pcolor ="green"
    else:
        pcolor = "blue"
    colorarr = np.append(colorarr, pcolor)
```

```python


def pearsonCoefficient(u, v):
    delta_u = u - np.mean(u)
    delta_v = v - np.mean(v)
    numerator = np.matmul(delta_u,delta_v.T)
    denominator = np.sqrt(np.matmul(delta_u,delta_u.T)*np.matmul(delta_v,delta_v.T))
    Pearson_correlation_coefficient = numerator/denominator
    print(Pearson_correlation_coefficient)
```

# Code

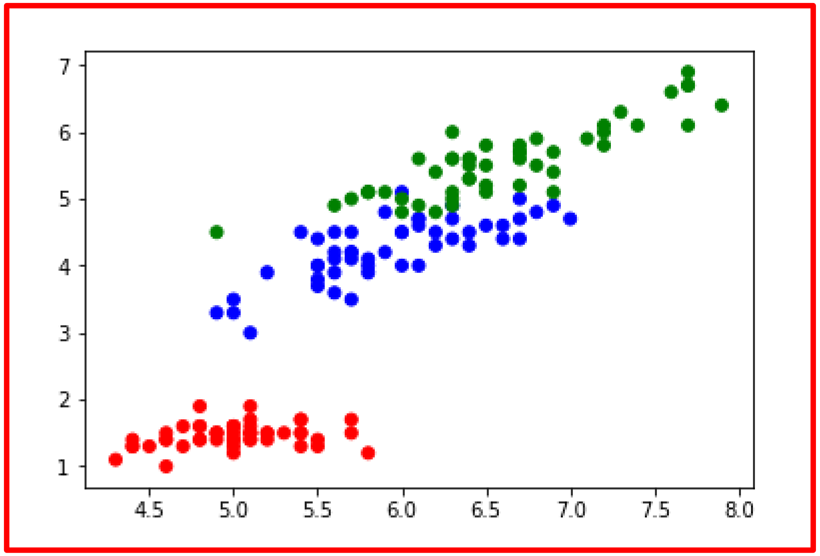
```
pearsonCoefficient(featureVec0, featureVec1)
plot.scatter(featureVec0, featureVec1, c = colorarr)

plot.figure()
pearsonCoefficient(featureVec0, featureVec2)
plot.scatter(featureVec0, featureVec2, c = colorarr)

plot.figure()
pearsonCoefficient(featureVec0, featureVec3)
plot.scatter(featureVec0, featureVec3, c = colorarr)

plot.figure()
pearsonCoefficient(featureVec1, featureVec2)
plot.scatter(featureVec1, featureVec2, c = colorarr)

plot.figure()
pearsonCoefficient(featureVec1, featureVec3)
plot.scatter(featureVec1, featureVec3, c = colorarr)

plot.figure()
pearsonCoefficient(featureVec2, featureVec3)
plot.scatter(featureVec2, featureVec3, c = colorarr)
```

# Correlation coefficient output using features 0 and 1 (Sepal length and width)



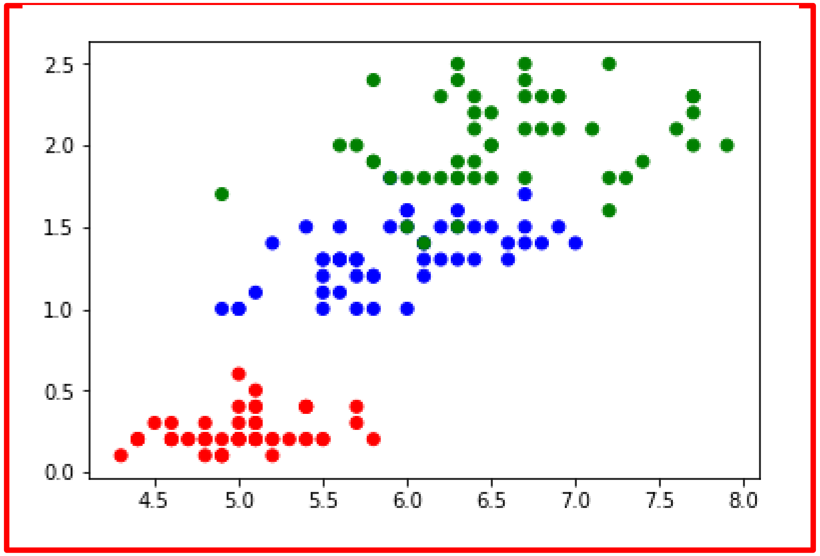

Pearson Correlation Coefficient = -0.10936924995064934

# Correlation coefficient output using features 0 and 2 (Sepal length and petal length)



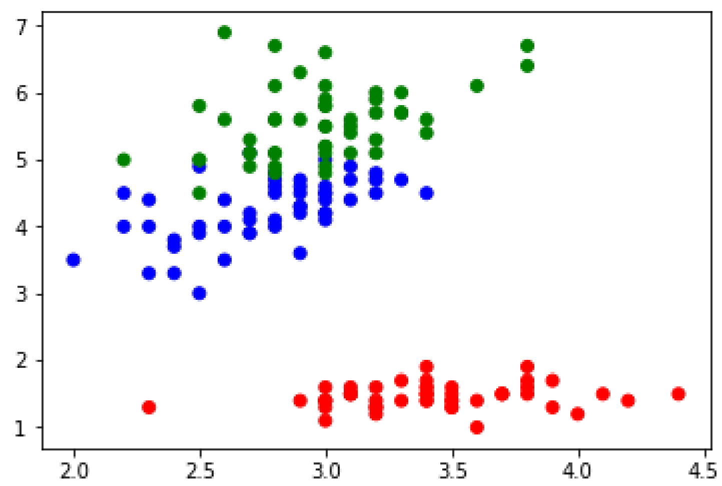

Pearson Correlation Coefficient = 0.8717541573048712

# Correlation coefficient output using features 0 and 3 (Sepal length and petal width)



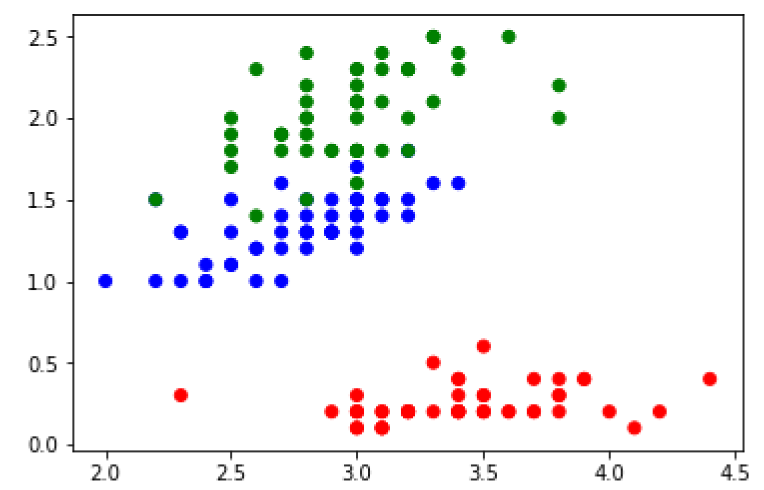

Pearson Correlation Coefficient = 0.8179536333691633

# Correlation coefficient output using features 1 and 2 (Sepal width and petal length)



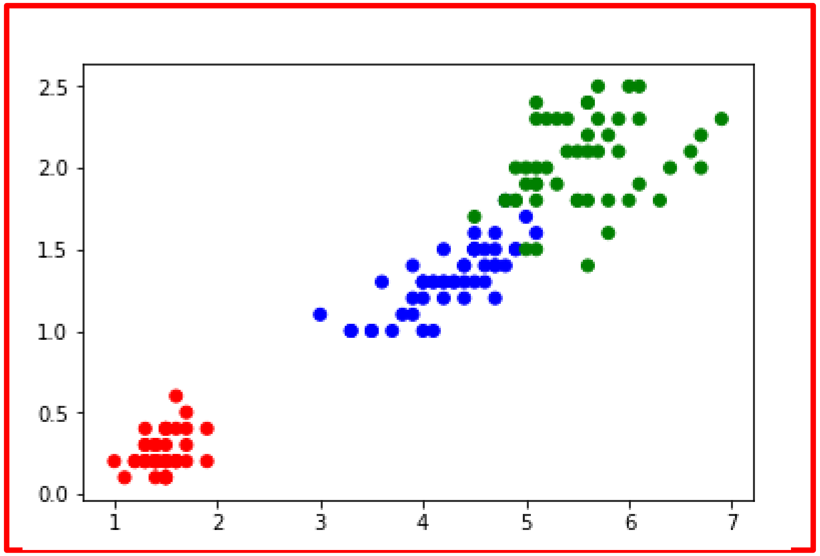

Pearson Correlation Coefficient = -0.42051609640115445

# Correlation coefficient output using features 1 and 3 (Sepal width and petal width)




Pearson Correlation Coefficient = -0.35654408961380585

# Correlation coefficient output using features 2 and 3 (Petal length and width)




Pearson Correlation Coefficient = 0.9627570970509662