Name: Ankit Sharma

Email: kumarankitx022@gmail.com

Mob no.: +91-7677241423

Week: 03

ASSIGNMENT – 06

TOPIC: MULTIPLE LINEAR REGRESSION (With Backward Elimination)

Q.1. Automatic Backward Elimination.

Ans. Backward elimination is a feature selection technique while building a machine learning model. It is used to remove those features that do not have a significant effect on the dependent variable or prediction of output.

STEPS FOR BACKWARD ELIMINATION:

Step-1: Firstly, We need to select a significance level to stay in the model. (SL=0.05)

Step-2: Fit the complete model with all possible predictors/independent variables.

Step-3: Choose the predictor which has the highest P-value, such that.

- **a.** If P-value >SL, go to step 4.
- **b.** Else Finish, and Our model is ready.

Step-4: Remove that predictor.

Step-5: Rebuild and fit the model with the remaining variables.

In [5]:

```
# Creating automated backward elimination function with p-values
import statsmodels.api as sm
def backwardElimination(x, SL):
    numVars = len(x[0])
    temp = np.zeros((50,6)).astype(int)
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(Y, x).fit()
        maxVar = max(regressor_OLS.pvalues).astype(float)
        if maxVar > SL:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) == maxVar):
                    temp[:,j] = x[:, j]
                    x = np.delete(x, j, 1)
                    tmp_regressor = sm.OLS(Y, x).fit()
    regressor_OLS.summary()
    return x

# Applying the backward elimination

X = np.append(arr = np.ones((50, 1)).astype(int), values = X, axis = 1)
SL = 0.05
X_opt = X[:,[0,1,2,3,4,5]]
X_opt = np.array(X_opt, dtype=float)

X_Res = backwardElimination(X_opt, SL)
regressor_OLS = sm.OLS(endog = Y, exog = X_Res).fit()
regressor_OLS.summary()
```

Out[5]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.947
Model:	OLS	**Adj. R-squared:**	0.945
Method:	Least Squares	**F-statistic:**	849.8
Date:	Tue, 19 May 2020	**Prob (F-statistic):**	3.50e-32
Time:	16:42:53	**Log-Likelihood:**	-527.44
No. Observations:	50	**AIC:**	1059.
Df Residuals:	48	**BIC:**	1063.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.903e+04	2537.897	19.320	0.000	4.39e+04	5.41e+04
x1	0.8543	0.029	29.151	0.000	0.795	0.913

Omnibus:	13.727	**Durbin-Watson:**	1.116
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	18.536
Skew:	-0.911	**Prob(JB):**	9.44e-05
Kurtosis:	5.361	**Cond. No.**	1.65e+05

ASSIGNMENT – 07

TOPIC: DECISION TREE

Q.1. Write notes on Decision Tree?

Ans. Decision tree is a type of supervised learning algorithm (having a predefined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables.

Types of Decision Trees

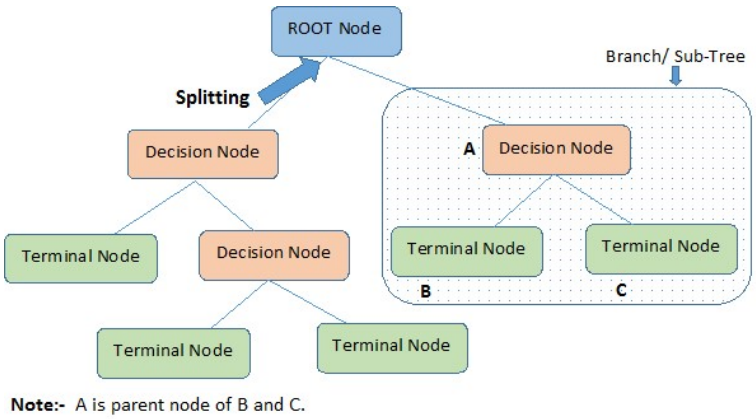
Types of decision tree is based on the type of target variable we have. It can be of two types:

1. **Categorical Variable Decision Tree:** Decision Tree which has categorical target variable then it called as categorical variable decision tree. Example: - In above scenario of student problem, where the target variable was "Student will play cricket or not" i.e. YES or NO.
2. **Continuous Variable Decision Tree:** Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Important Terminology related to Tree based Algorithms

Let's look at the basic terminology used with Decision trees:

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.

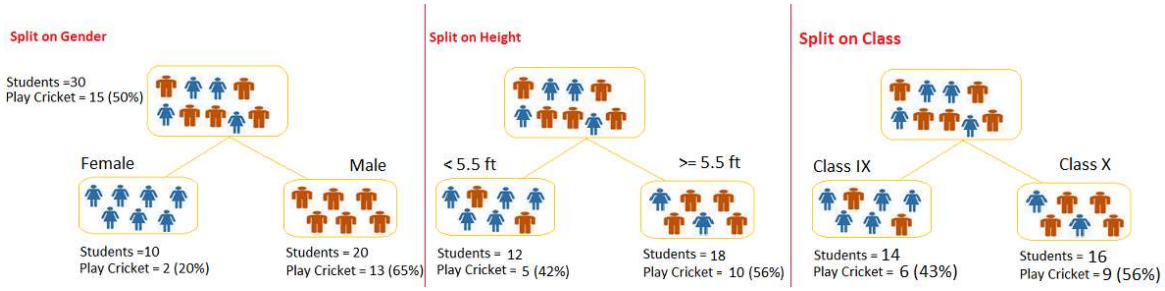


Note:- A is parent node of B and C.

Example:-

Let's say we have a sample of 30 students with three variables Gender (Boy/ Girl), Class (IX/ X) and Height (5 to 6 ft.). 15 out of these 30 play cricket in leisure time. Now, I want to create a model to predict who will play cricket during leisure period? In this problem, we need to segregate students who play cricket in their leisure time based on highly significant input variable among all three.

This is where decision tree helps, it will segregate the students based on all values of three variable and identify the variable, which creates the best homogeneous sets of students (which are heterogeneous to each other). In the snapshot below, you can see that variable Gender is able to identify best homogeneous sets compared to the other two variables.



As mentioned above, decision tree identifies the most significant variable and its value that gives best homogeneous sets of population. Now the question which arises is, how does it identify the variable and the split? To do this, decision tree uses various algorithms, which we will discuss in the following section.

How does a tree based algorithms decide where to split?

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria is different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable. Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

Gini

Gini says, if we select two items from a population at random then they must be of same class and probability for this is 1 if population is pure.

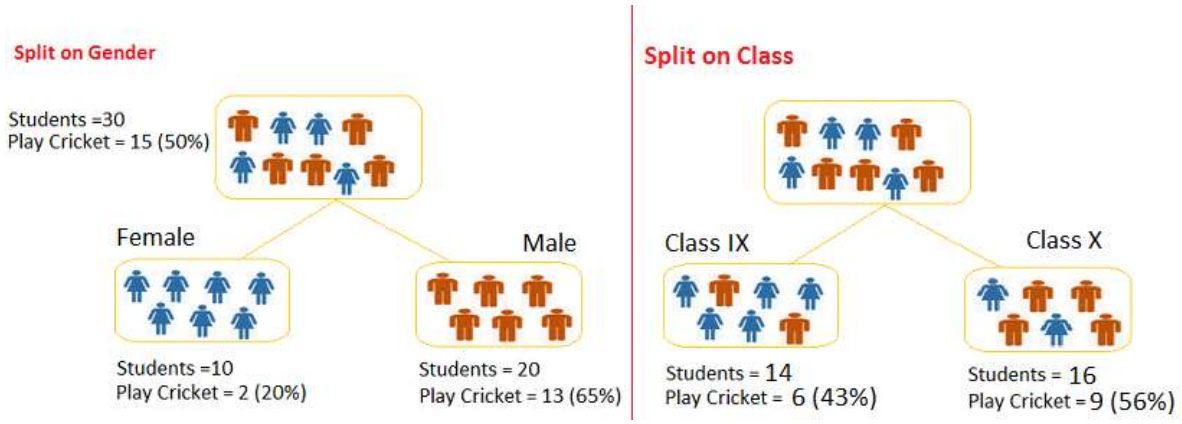
1. It works with categorical target variable "Success" or "Failure".
2. It performs only Binary splits

3. Higher the value of Gini higher the homogeneity.
4. CART (Classification and Regression Tree) uses Gini method to create binary splits.

Steps to Calculate Gini for a split

1. Calculate Gini for sub-nodes, using formula sum of square of probability for success and failure (p^2+q^2).
2. Calculate Gini for split using weighted Gini score of each node of that split

Example: – Referring to example used above, where we want to segregate the students based on target variable (playing cricket or not). In the snapshot below, we split the population using two input variables Gender and Class. Now, I want to identify which split is producing more homogeneous sub-nodes using Gini.



Split on Gender:

1. Calculate, Gini for sub-node Female = (0.2)*(0.2)+(0.8)*(0.8)=0.68
2. Gini for sub-node Male = (0.65)*(0.65)+(0.35)*(0.35)=0.55
3. Calculate weighted Gini for Split Gender = (10/30)*0.68+(20/30)*0.55 = **0.59**

Similar for Split on Class:

1. Gini for sub-node Class IX = (0.43)*(0.43)+(0.57)*(0.57)=0.51
2. Gini for sub-node Class X = (0.56)*(0.56)+(0.44)*(0.44)=0.51
3. Calculate weighted Gini for Split Class = (14/30)*0.51+(16/30)*0.51 = **0.51**

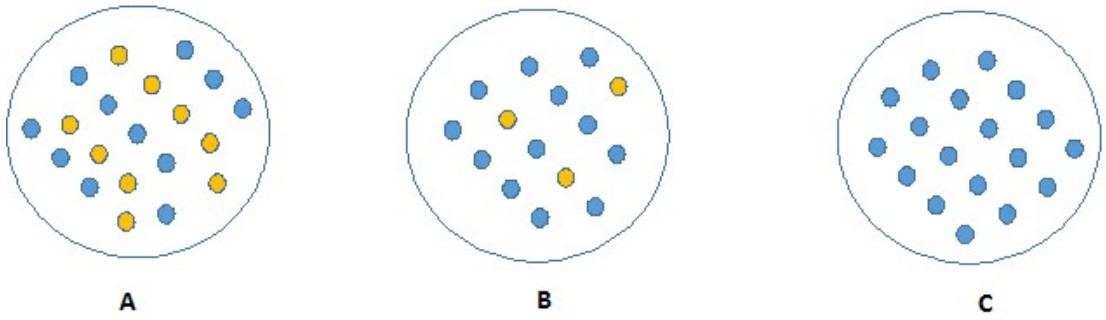
Above, you can see that Gini score for *Split on Gender* is higher than *Split on Class,* hence, the node split will take place on Gender.

You might often come across the term 'Gini Impurity' which is determined by subtracting the Gini value from 1. So mathematically we can say,

Gini Impurity = 1-Gini

Information Gain:

Look at the image below and think which node can be described easily. I am sure, your answer is C because it requires less information as all values are similar. On the other hand, B requires more information to describe it and A requires the maximum information. In other words, we can say that C is a pure node, B is less Impure and A is more impure.



Now, we can build a conclusion that less impure node requires less information to describe it. And, more impure node requires more information. Information theory is a measure to define this degree of disorganization in a system known as Entropy. If the sample is completely homogeneous, then the entropy is zero and if the sample is an equally divided (50% – 50%), it has entropy of one.

Entropy can be calculated using formula:- $\text{Entropy} = -p \log_2 p - q \log_2 q$

Here p and q is probability of success and failure respectively in that node. Entropy is also used with categorical target variable. It chooses the split which has lowest entropy compared to parent node and other splits. The lesser the entropy, the better it is.

Steps to calculate entropy for a split:

1. Calculate entropy of parent node
2. Calculate entropy of each individual node of split and calculate weighted average of all sub-nodes available in split.

Example: Let's use this method to identify best split for student example.

1. Entropy for parent node = - (15/30) log2 (15/30) – (15/30) log2 (15/30) = **1**. Here 1 shows that it is an impure node.
2. Entropy for Female node = -(2/10) log2 (2/10) – (8/10) log2 (8/10) = 0.72 and for male node, -(13/20) log2 (13/20) – (7/20) log2 (7/20) = **0.93**
3. Entropy for split Gender = Weighted entropy of sub-nodes = (10/30)*0.72 + (20/30)*0.93 = **0.86**
4. Entropy for Class IX node, - (6/14) log2 (6/14) – (8/14) log2 (8/14) = 0.99 and for Class X node, -(9/16) log2 (9/16) – (7/16) log2 (7/16) = 0.99.
5. Entropy for split Class = (14/30)*0.99 + (16/30)*0.99 = **0.99**

Above, you can see that entropy for *Split on Gender* is the lowest among all, so the tree will split on *Gender*. We can derive information gain from entropy as **1- Entropy.**

Reduction in Variance

Till now, we have discussed the algorithms for categorical target variable. Reduction in variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with lower variance is selected as the criteria to split the population:

$$\text{Variance} = \frac{\Sigma(X - \overline{X})^2}{n}$$

Above X-bar is mean of the values, X is actual and n is number of values.

Steps to calculate Variance:

1. Calculate variance for each node.
2. Calculate variance for each split as weighted average of each node variance.

Example: - Let's assign numerical value 1 for play cricket and 0 for not playing cricket. Now follow the steps to identify the right split:

1. Variance for Root node, here mean value is (15*1 + 15*0)/30 = 0.5 and we have 15 one and 15 zero. Now variance would be ((1-0.5)^2+(1-0.5)^2+….15 times+(0-0.5)^2+(0-0.5)^2+…15 times) / 30, this can be written as (15*(1-0.5)^2+15*(0-0.5)^2) / 30 = **0.25**
2. Mean of Female node = (2*1+8*0)/10=0.2 and Variance = (2*(1-0.2)^2+8*(0-0.2)^2) / 10 = 0.16
3. Mean of Male Node = (13*1+7*0)/20=0.65 and Variance = (13*(1-0.65)^2+7*(0-0.65)^2) / 20 = 0.23
4. Variance for Split Gender = Weighted Variance of Sub-nodes = (10/30)*0.16 + (20/30) *0.23 = **0.21**
5. Mean of Class IX node = (6*1+8*0)/14=0.43 and Variance = (6*(1-0.43)^2+8*(0-0.43)^2) / 14= 0.24
6. Mean of Class X node = (9*1+7*0)/16=0.56 and Variance = (9*(1-0.56)^2+7*(0-0.56)^2) / 16 = 0.25
7. Variance for Split Gender = (14/30)*0.24 + (16/30) *0.25 = **0.25**

Above, you can see that Gender split has lower variance compare to parent node, so the split would take place on *Gender* variable.

ASSIGNMENT – 08

TOPIC: RANDOM FOREST

Q.1. Check for MAE, MSE, R2-SCORE, RMSE based on different n_estimators.

Q.2. Check for minimum RMSE and returning best n_estimator.

Ans. 1)

1. **Mean absolute error:** It is the mean of the absolute value of the errors. This is the easiest of the metrics to understand since it's just average error.

2. **Mean Squared Error (MSE):** MSE is the mean of the squared error. It's more popular than Mean absolute error because the focus is geared more towards large errors. This is due to the squared term exponentially increasing larger errors in comparison to smaller ones.

3. **Root Mean Squared Error (RMSE):** RMSE is the standard deviation of the errors which occur when a prediction is made on a dataset. This is the same as MSE (Mean Squared Error) but the root of the value is considered while determining the accuracy of the model.

4. **R2-Score:** R-squared is not error, but is a popular metric for accuracy of your model. It represents how close the data are to the fitted regression line. **The higher the R-squared, the better the model fits your data. Best possible score is 1.0 and it can be negative** (because the model can be arbitrarily worse).

5. **n_estimators:** The number of trees in the forest. Since **Random Forest** is an ensemble method comprising of creating multiple **decision** trees, this parameter is used to control the number of trees to be used in the process.

Check for MAE, MSE, R2-score, RMSE based on different n_estimators

In [13]:

```
#Check for MAE, MSE, R2-score, RMSE based on different n_estimators
import random as rd
from sklearn.metrics import r2_score, mean_squared_error
i = rd.sample(range(100),10)
x,a,b,c,d = [],[],[],[],[]
for e in i:
    from sklearn.ensemble import RandomForestRegressor
    reg = RandomForestRegressor(n_estimators=e, random_state = 0)
    reg.fit(X,y)
    pred = reg.predict(X)
    x.append(e)
    a.append(np.mean(np.absolute(pred - y)).round(decimals = 2))
    b.append(np.sqrt(mean_squared_error(y, pred)).round(decimals = 2))
    c.append(r2_score(pred , y).round(decimals = 2))
    d.append(np.sqrt(np.mean((pred - y) ** 2)).round(decimals = 2))

    rand_check = pd.DataFrame({
        'n_estimators': np.array(x).flatten(),
        'MAE': np.array(a).flatten(),
        'MSE': np.array(b).flatten(),
        'R2-Score': np.array(c).flatten(),
        'RMSE': np.array(d).flatten(),
    })
rand_check
```

Out[13]:

	n_estimators	MAE	MSE	R2-Score	RMSE
0	35	26814.29	73346.44	0.89	73346.44
1	9	33000.00	60696.57	0.93	60696.57
2	80	26437.50	65478.66	0.92	65478.66
3	81	27283.95	67474.30	0.91	67474.30
4	36	26430.56	71351.60	0.90	71351.60
5	60	25658.33	66271.58	0.92	66271.58
6	63	25682.54	65660.94	0.92	65660.94
7	75	26073.33	66626.38	0.92	66626.38
8	91	26159.34	69082.49	0.91	69082.49
9	95	26915.79	71190.65	0.90	71190.65

Check for minimum RMSE and returning best best n_estimator

In [14]:

```
#Check for minimum RMSE and returning best best n_estimator
#p = np.where(d == np.amin(d))
p= d.index(min(d))
print("Best n_estimator value: ", x[p])
evaluation = pd.DataFrame({
    'n_estimators': np.array(x[p]).flatten(),
    'MAE': np.array(a[p]).flatten(),
    'MSE': np.array(b[p]).flatten(),
    'R2-Score': np.array(c[p]).flatten(),
    'RMSE': np.array(d[p]).flatten(),
})
evaluation
```

```
Best n_estimator value:  9
```

Out[14]:

	n_estimators	MAE	MSE	R2-Score	RMSE
0	9	33000.0	60696.57	0.93	60696.57

ASSIGNMENT – 09

TOPIC: SUPPORT VECTOR MACHINE (SVM)

Q.1. Write notes for sklearn.svm.SVC class focusing more on kernel values – 'linear', 'rbf', 'poly', etc.

Q.2. Write notes for matplotlib library focus on different plots. Ex. 'contourf'.

Q.3. Write notes about numpy – 'meshgrid', 'ravel'.

Ans. 1) sklearn.svm.SVC:

class `sklearn.svm.`**SVC**(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*)

SVM Kernels

The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. In other words, you can say that it converts non-separable problem to separable problems by adding more dimension to it. It is most useful in non-linear separation problem. Kernel trick helps you to build a more accurate classifier.

- **Linear Kernel:** A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.

K(x, xi) = sum(x * xi)

- **Polynomial Kernel:** A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.

K(x, xi) = 1 + sum(x * xi)^d

Where d is the degree of the polynomial. d=1 is similar to the linear transformation. The degree needs to be manually specified in the learning algorithm.

- **Radial Basis Function Kernel:** The Radial basis function kernel is a popular kernel function commonly used in support vector machine classification. RBF can map an input space in infinite dimensional space.

K(x, xi) = exp(-gamma * sum(x-xi^2))

Here gamma is a parameter, which ranges from 0 to 1. A higher value of gamma will perfectly fit the training dataset, which causes over-fitting. Gamma=0.1 is considered to be a good default value. The value of gamma needs to be manually specified in the learning algorithm.

Other basic parameters in sklearn.svm.SVC class:

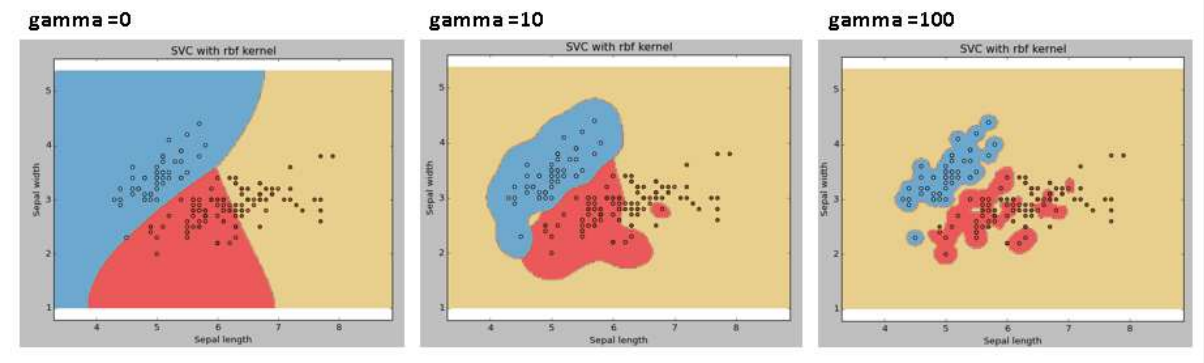
```
svc = svm.SVC(kernel='rbf', C=1,gamma=0)
```

- **Gamma:** Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.
- **C:** Penalty parameter C of the error term. It also controls the trade-off between smooth decision boundaries and classifying the training points correctly.

Example: Comparison on different values of gamma and C using iris dataset.

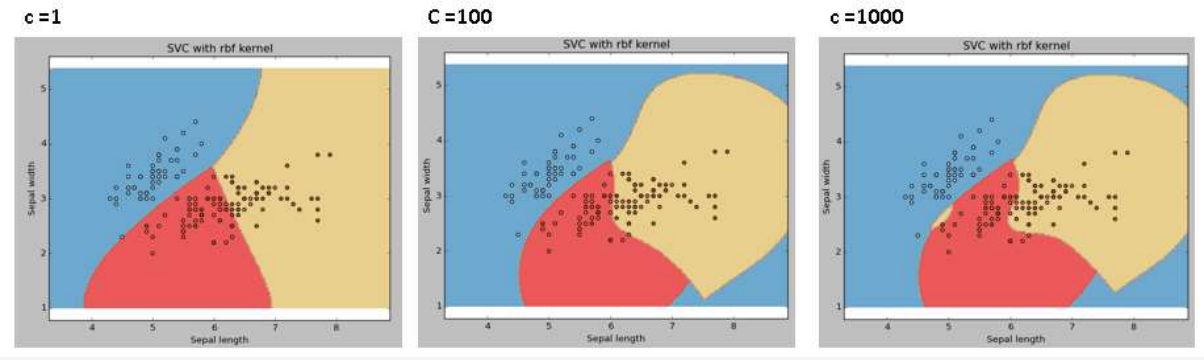
```
1. Difference when using different values of gamma in the above code snippet.

Kernel:  rbf                    rbf                    rbf
gamma:   0                      10                     100
C:       1                      1                      1
```

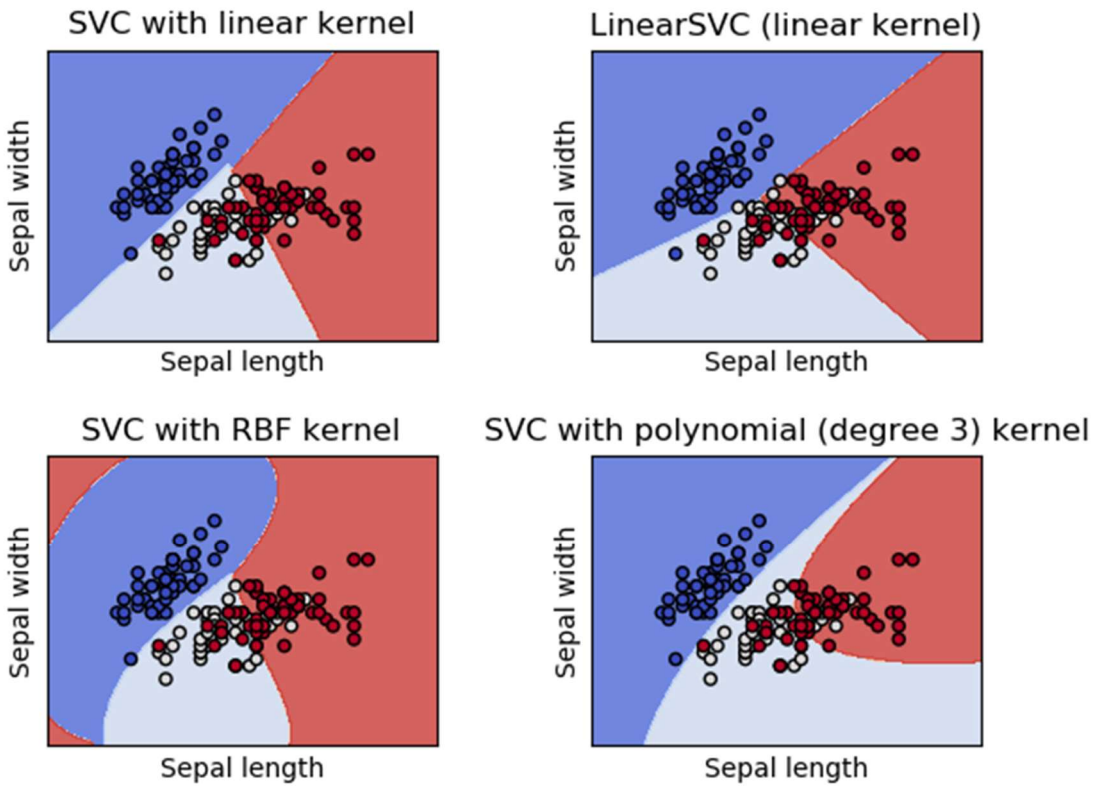



```
2. Difference when using different values of C in the above code snippet.

Kernel:  rbf                    rbf                    rbf
gamma:   0                      10                     100
C:       1                      100                    1000
```



Comparison of different linear SVM classifiers on a 2D projection of the iris dataset using different kernels (source sklearn.org):



Ans. 3) Numpy functions meshgrid and ravel:

meshgrid: numpy.meshgrid(*xi, **kwargs)

The numpy.meshgrid function is used to create a rectangular grid out of two given one-dimensional arrays representing the Cartesian indexing or Matrix indexing.

- Return coordinate matrices from coordinate vectors.
- Make N-D coordinate arrays for vectorised evaluations of N-D scalar/vector fields over N-D grids, given one-dimensional coordinate arrays x1, x2… xn.

Parameters:

- **x1, x2,…, xn (array_like):** This parameter defines the 1-dimensional array, which represents the coordinates of a grid.
- **indexing: {'xy', 'ij'}(optional):** This is an optional argument which defines the Cartesian 'xy'(by default) or matrix ('ij') indexing of output.
- **sparse: bool(optional)**: This parameter is also optional. If we need a sparse grid for conserving memory, we have to set this parameter to True. By default, it is set to False.
- **copy: bool(optional)**: The aim of this optional argument is that it returns a copy of the original array for conserving memory. By default, it is set to False.

- If both **sparse** and **copy** parameters are set to False, then it will return non-contiguous arrays. In addition, more than one element of a broadcast array can refer to a single memory location. If we need to write into the arrays, then we have to make copies first.

Returns: (x1, x2, …, xn) The coordinate length from the coordinate vector is returned from this function.

ravel: `numpy.ravel(a, order='C')`

The numpy module of Python provides a function called numpy.ravel, which is used to change a 2-dimensional array or a multi-dimensional array into a contiguous flattened array. The returned array has the same data type as the source array or input array. If the input array is a masked array, the returned array will also be a masked array.

- Return a contiguous flattened array.
- A 1-D array, containing the elements of the input, is returned. A copy is made only if needed.

Parameters:

- **a (array_like):** This parameter defines the input array, which we want to change in a contiguous flattened array. The array elements are read in the order specified by the order parameter and packed as a 1-D array.
- **order: {'C','F', 'A', 'K'}(optional):** If we set the order parameter to 'C', it means that the array gets flattened in row-major order. If 'F' is set, the array gets flattened in column-major order. The array is flattened in column-major order only when 'A' is Fortran contiguous in memory, and when we set the order parameter to 'A'. The last order is 'K', which flatten the array in same order in which the elements occurred in the memory. By default, this parameter is set to 'C'.

Returns: This function returns a contiguous flatten array with the same data type as an input array and has shape equal to (**x.size**).

Meshgrid example

In [1]:

```
import numpy as np
na, nb = (5, 3)
a = np.linspace(1, 2, na)
b = np.linspace(1, 2, nb)
xa, xb = np.meshgrid(a, b)
```

In [2]:

```
xa
```

Out[2]:

```
array([[1.  , 1.25, 1.5 , 1.75, 2.  ],
       [1.  , 1.25, 1.5 , 1.75, 2.  ],
       [1.  , 1.25, 1.5 , 1.75, 2.  ]])
```

In [3]:

```
xb
```

Out[3]:

```
array([[1. , 1. , 1. , 1. , 1. ],
       [1.5, 1.5, 1.5, 1.5, 1.5],
       [2. , 2. , 2. , 2. , 2. ]])
```

Ravel example

In [4]:

```
import numpy as np
x = np.array([[1, 3, 5], [11, 35, 56]])
y = np.ravel(x, order='F')
z = np.ravel(x, order='C')
p = np.ravel(x, order='A')
q = np.ravel(x, order='K')
```

In [5]:

```
print("y: ", y)
print("z: ", z)
print("p: ", p)
print("q: ", q)
```

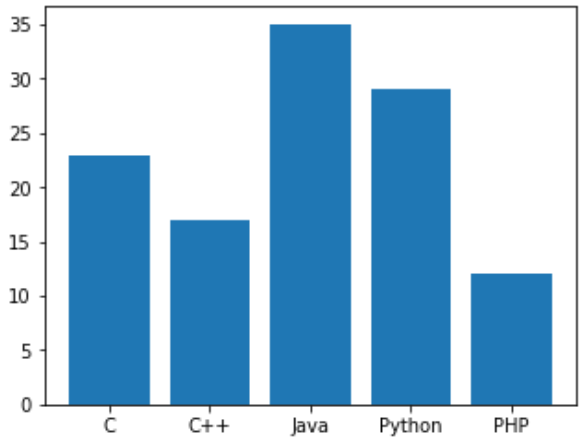
```
y:  [ 1 11  3 35  5 56]
z:  [ 1  3  5 11 35 56]
p:  [ 1  3  5 11 35 56]
q:  [ 1  3  5 11 35 56]
```

Ans 2.) Different plots using matplotlib library

1. Bar Plot: A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.

In [12]:

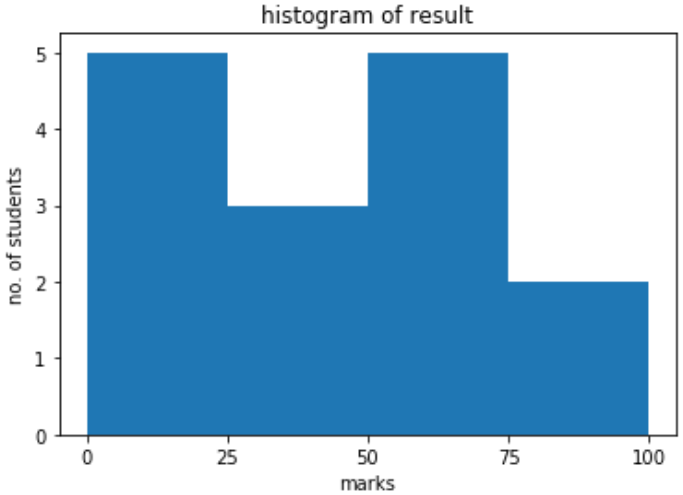
```
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(4, 3))
ax=fig.add_axes([0,0,1,1])
langs=['C', 'C++', 'Java', 'Python', 'PHP']
students=[23,17,35,29,12]
ax.bar(langs,students)
plt.show()
```



2. Histogram: A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable. It is a kind of bar graph.

In [2]:

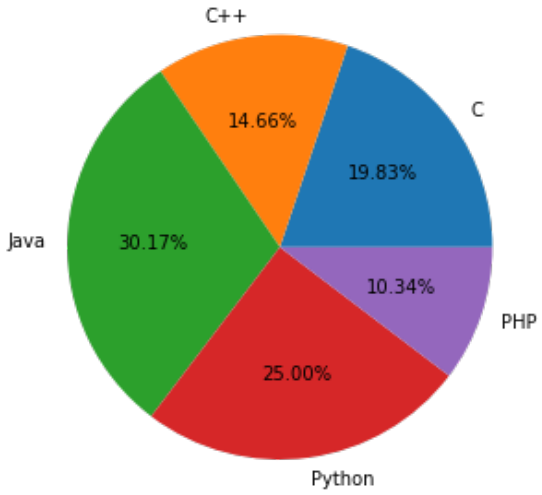
```
import numpy as np
fig,ax=plt.subplots(1,1)
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,25,50,75,100])
ax.set_title("histogram of result")
ax.set_xticks([0,25,50,75,100])
ax.set_xlabel('marks')
ax.set_ylabel('no. of students')
plt.show()
```



3. Pie Chart: **A Pie Chart can only display one series of data. Pie charts show the size of items (called wedge) in one data series, proportional to the sum of the items. The data points in a pie chart are shown as a percentage of the whole pie.**

In [3]:

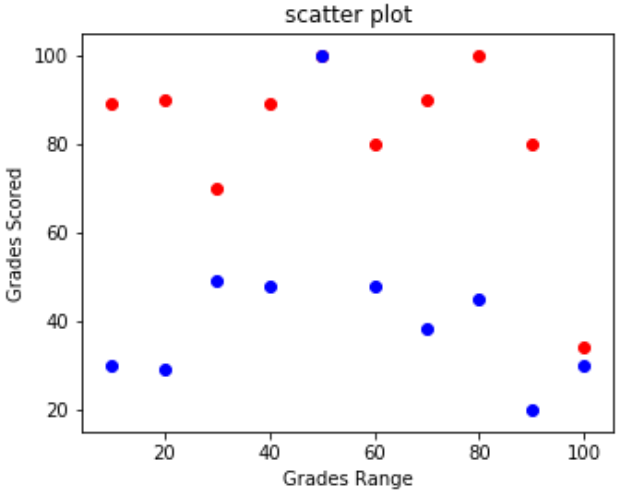
```
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.axis('equal')
langs=['C', 'C++', 'Java', 'Python', 'PHP']
students=[23,17,35,29,12]
ax.pie(students, labels=langs,autopct='%1.2f%%')
plt.show()
```



4. Scatter plot: **Scatter plots are used to plot data points on horizontal and vertical axis in the attempt to show how much one variable is affected by another. Each row in the data table is represented by a marker the position depends on its values in the columns set on the X and Y axes. A third variable can be set to correspond to the color or size of the markers, thus adding yet another dimension to the plot**

In [13]:

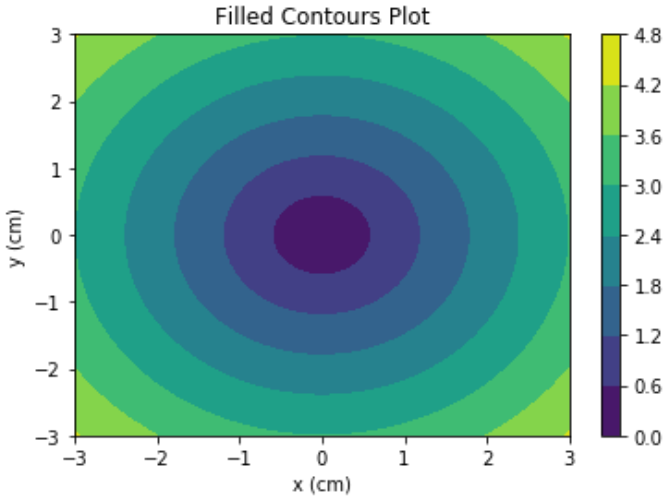
```
girls_grades = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys_grades = [30, 29, 49, 48, 100, 48, 38, 45, 20, 30]
grades_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
fig=plt.figure(figsize=(4, 3))
ax=fig.add_axes([0,0,1,1])
ax.scatter(grades_range, girls_grades, color='r')
ax.scatter(grades_range, boys_grades, color='b')
ax.set_xlabel('Grades Range')
ax.set_ylabel('Grades Scored')
ax.set_title('scatter plot')
plt.show()
```



5. Contour plot: Contour plots (sometimes called Level Plots) are a way to show a three-dimensional surface on a two-dimensional plane. It graphs two predictor variables X Y on the y-axis and a response variable Z as contours. A contour plot is appropriate if you want to see how alue Z changes as a function of two inputs X and Y, such that Z = f(X,Y). A contour line or isoline of a function of two variables is a curve along which the function has a constant value. The independent variables x and y are usually restricted to a regular grid called meshgrid. The numpy.meshgrid creates a rectangular grid out of an array of x values and an array of y values.

In [5]:

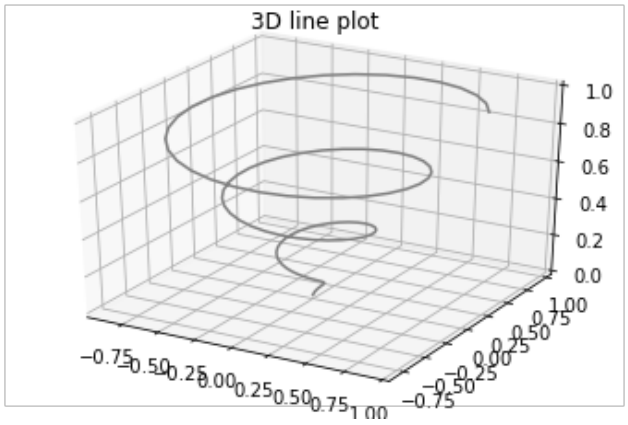
```
xlist = np.linspace(-3.0, 3.0, 100)
ylist = np.linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = np.sqrt(X**2 + Y**2)
fig,ax=plt.subplots(1,1)
cp = ax.contourf(X, Y, Z)
fig.colorbar(cp) # Add a colorbar to a plot
ax.set_title('Filled Contours Plot')
ax.set_xlabel('x (cm)')
ax.set_ylabel('y (cm)')
plt.show()
```



6. 3D plot: Three-dimensional plots are enabled by importing the mplot3d toolkit, included with the Matplotlib package. A three-dimensional axes can be created by passing the keyword projection='3d' to any of the normal axes creation routines.

In [6]:

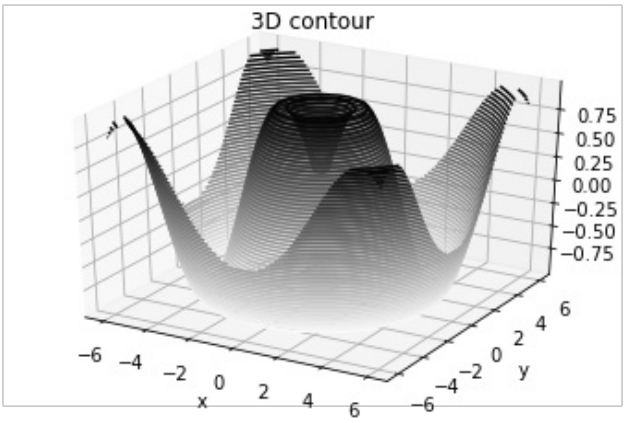
```
from mpl_toolkits import mplot3d
fig = plt.figure()
ax = plt.axes(projection='3d')
z = np.linspace(0, 1, 100)
x = z * np.sin(20 * z)
y = z * np.cos(20 * z)
ax.plot3D(x, y, z, 'gray')
ax.set_title('3D line plot')
plt.show()
```



7. 3D Contour Plot: The ax.contour3D() function creates three-dimensional contour plot. It requires all the input data to be in the form of two-dimensional regular grids, with the Z-data evaluated at each point. Here, we will show a three-dimensional contour diagram of a threedimensional sinusoidal function.

In [19]:

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))
x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x'),ax.set_ylabel('y'), ax.set_zlabel('z')
ax.set_title('3D contour')
plt.show()
```



8. 3D Surface Plot: Surface plot shows a functional relationship between a designated dependent variable (Y), and two independent variables (X and Z). The plot is a companion plot to the contour plot. A surface plot is like a wireframe plot, but each face of the wireframe is a filled polygon. This can aid perception of the topology of the surface being visualized. The plot_surface() function x,y and z as arguments.

In [18]:

```
x = np.outer(np.linspace(-2, 2, 30), np.ones(30))
y = x.copy().T # transpose
z = np.cos(x ** 2 + y ** 2)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(x, y, z,cmap='viridis', edgecolor='none')
ax.set_title('Surface plot')
ax.set_xlabel('x'),ax.set_ylabel('y'),ax.set_zlabel('z')
plt.show()
```