In [8]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#from sklearn.preprocessing import Imputer
from sklearn.impute import SimpleImputer

dataset=pd.read_csv('Data.csv')
dataset
X=dataset.iloc[:,:-1].values
X
Y=dataset.iloc[:, 3].values
Y
```

Out[8]:

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

In [4]:

```
import sklearn
print (sklearn.__version__)
```

```
0.22.2.post1
```

In [6]:

```
np.shape(X)
```

Out[6]:

```
(10, 3)
```

In [7]:

```
print(X)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

In [26]:

```
imputer=SimpleImputer(missing_values='NaN',strategy='mean')
imputer.fit(X[:,1:3])
X[:,1:3]=imputer.transform(X[:,1:3])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-26-547a83132c8e> in <module>()
      1 imputer=SimpleImputer(missing_values='NaN',strategy='mean')
----> 2 imputer.fit(X[:,1:3])
      3 X[:,1:3]=imputer.transform(X[:,1:3])

/usr/local/lib/python3.6/dist-packages/sklearn/impute/_base.py in fit(self, X, y)
    266         self : SimpleImputer
    267         """
--> 268         X = self._validate_input(X)
    269         super()._fit_indicator(X)
    270
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/impute/_base.py in _validate_input(self, X
)
    240                 raise new_ve from None
    241             else:
--> 242                 raise ve
    243
    244         _check_inputs_dtype(X, self.missing_values)

/usr/local/lib/python3.6/dist-packages/sklearn/impute/_base.py in _validate_input(self, X
)
    233         try:
    234             X = check_array(X, accept_sparse='csc', dtype=dtype,
--> 235                             force_all_finite=force_all_finite, copy=self.copy)
    236         except ValueError as ve:
    237             if "could not convert" in str(ve):

/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py in check_array(array,
accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allo
w_nd, ensure_min_samples, ensure_min_features, warn_on_dtype, estimator)
    576         if force_all_finite:
    577             _assert_all_finite(array,
--> 578                                allow_nan=force_all_finite == 'allow-nan')
    579
    580     if ensure_min_samples > 0:

/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py in _assert_all_finite(
X, allow_nan, msg_dtype)
     58                     msg_err.format
     59                     (type_err,
---> 60                      msg_dtype if msg_dtype is not None else X.dtype)
     61             )
     62     # for object dtype data, we only check for NaNs (GH-13254)

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

In [12]:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

labelencoder_X=LabelEncoder()
X[:,0]=labelencoder_X.fit_transform(X[:,0])

onehotencoder=OneHotEncoder(categorical_features=[0])
X=onehotencoder.fir_transform(X).array()
X.astype(int)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-208b5687142b> in <module>()
      4 X[:,0]=labelencoder_X.fit_transform(X[:,0])
      5
----> 6 onehotencoder=OneHotEncoder(categorical_features=[0])
      7 X=onehotencoder.fir_transform(X).array()
      8 X.astype(int)

TypeError: __init__() got an unexpected keyword argument 'categorical_features'
```

In [13]:

```
labelencoder_Y=LabelEncoder()
Y=labelencoder_Y.fit_transform(Y)
```

In [18]:

```
#or model selection
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, random_state=0)
```

In [19]:

```
X_train.astype(int)
```

```
X_test.astype(int)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-066cecc43ef8> in <module>()
----> 1 X_train.astype(int)
      2 X_test.astype(int)

ValueError: cannot convert float NaN to integer
```

In [24]:

```
from sklearn.preprocessing import StandardScaler
sc_X=StandardScaler()
X_train=sc_X.fit_transform(X_train)
X_test=sc_X.tansform(X_test)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-24-2d775a7aa79b> in <module>()
      2 sc_X=StandardScaler()
      3 X_train=sc_X.fit_transform(X_train)
----> 4 X_test=sc_X.tansform(X_test)

AttributeError: 'StandardScaler' object has no attribute 'tansform'
```

In [ ]:

```

```

I. Download iris dataset form UCI machine learning repository and use the dataset to develop the following classifiers and find the accuracy of the model. Compare and comment on the results:

1.Decision Tree Classifier

2.Naïve Bayes Classifier

3.Logistic Regression classifier

4.K-NN classifier

5.Logistic Regression model and apply PAC

6.Random Forest Model

7.Ada Boost Model

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

1. Decision Tree Classifier

In [3]:
```
dataset = pd.read_csv('Iris.csv')
dataset.head()
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [4]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [5]:
```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [6]:
```
# Lable Encoding the categorical data (dependent(y))
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
```

In [7]:
```
# Test Train Split
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, random_state
```

In [8]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [9]:
```
# Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

classifier_gini = DecisionTreeClassifier(criterion = 'gini', random_state = 0)
classifier_gini.fit(X_train, y_train)
```

Out[9]:
```
DecisionTreeClassifier(random_state=0)
```

In [10]:
```
y_pred = classifier.predict(X_test)
y_pred_gini = classifier_gini.predict(X_test)
```

In [11]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, plot_confusion_matrix
```

In [12]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm ,'\n')
prediction = accuracy_score(y_test, y_pred)
print(prediction * 100)
```

```
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]

100.0
```

In [13]:
```
cm_gini = confusion_matrix(y_test, y_pred)
print(cm_gini ,'\n')
prediction_gini = accuracy_score(y_test, y_pred)
print(prediction_gini * 100)
```

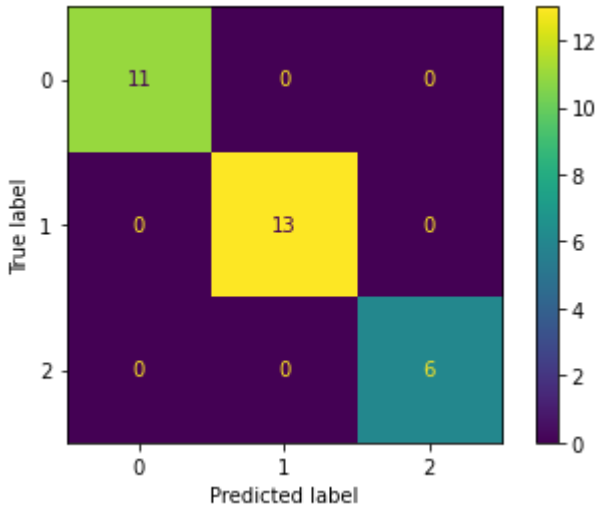
```
[[11  0  0]
 [ 0 13  0]
```

```
 [ 0  0  6]]

100.0
```

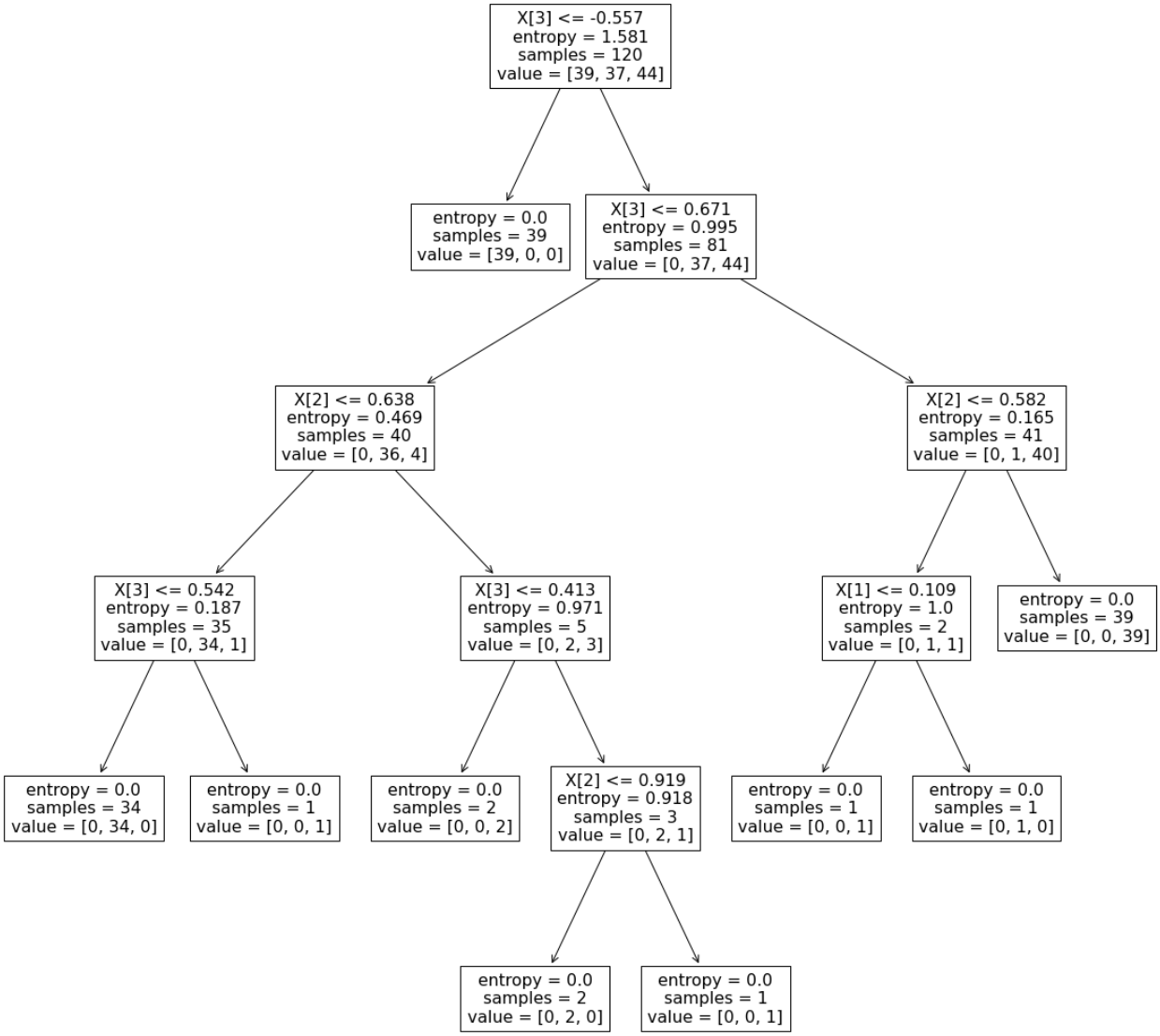
In [14]:

```
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



In [15]:

```
from sklearn import tree
plt.figure(figsize = (20, 20))
tree.plot_tree(classifier);
```



2. Naive Bayes Classifier

In [16]:
```
from sklearn.naive_bayes import GaussianNB
classifier_NB = GaussianNB()
classifier_NB.fit(X_train, y_train)
```

Out[16]:
```
GaussianNB()
```

In [17]:
```
y_pred_NB = classifier_NB.predict(X_test)
```

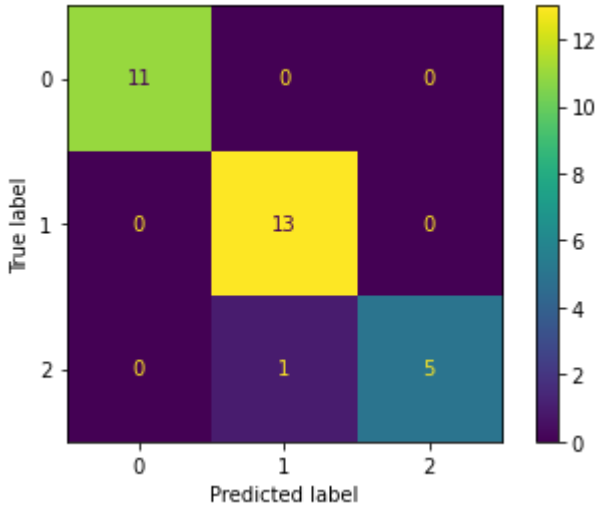
In [18]:
```
cm_NB = confusion_matrix(y_test, y_pred_NB)
print(cm_NB ,'\n')

prediction_NB = accuracy_score(y_test, y_pred_NB)
print('Accuracy is :',
      prediction_NB * 100)
```

```
[[11  0  0]
 [ 0 13  0]
 [ 0  1  5]]

Accuracy is : 96.66666666666667
```

In [19]:
```
plot_confusion_matrix(classifier_NB, X_test, y_test)
plt.show()
```



3. Logistic Regression classifier

In [20]:
```
from sklearn.linear_model import LogisticRegression
classifier_LR = LogisticRegression(random_state = 0)
classifier_LR.fit(X_train, y_train)
```

Out[20]:
```
LogisticRegression(random_state=0)
```

In [21]:
```
y_pred_LR = classifier_LR.predict(X_test)
```

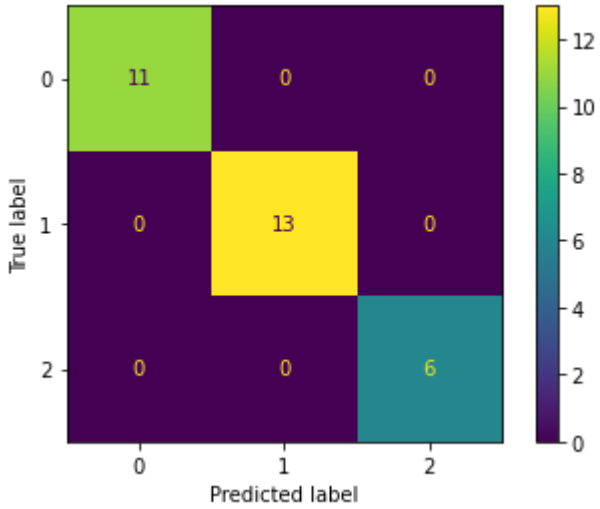
In [22]:
```
cm_LR = confusion_matrix(y_test, y_pred_LR)
print(cm_LR ,'\n')

prediction_LR = accuracy_score(y_test, y_pred_LR)
print('Accuracy is :', prediction_LR * 100)
```

```
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]

Accuracy is : 100.0
```

In [23]:
```
plot_confusion_matrix(classifier_LR, X_test, y_test)
plt.show()
```



4. K-NN classifier

In [24]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier_KNN = KNeighborsClassifier(n_neighbors = 3,
                                      metric = 'minkowski',
                                     p = 2)
classifier_KNN.fit(X_train, y_train)
```

Out[24]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [25]:
```
y_pred_KNN = classifier_KNN.predict(X_test)
```

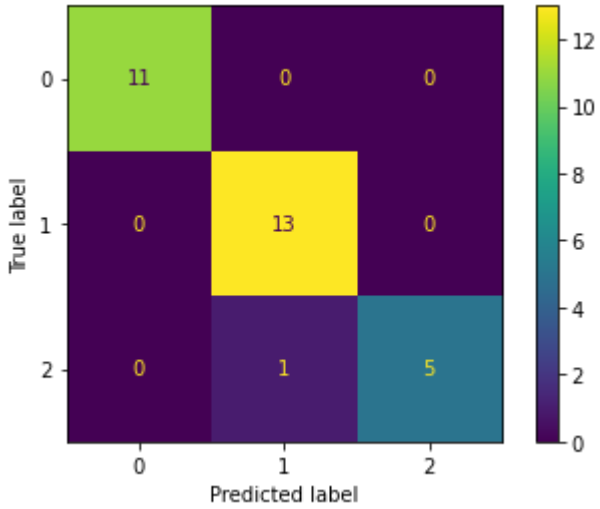
In [26]:
```
cm_KNN = confusion_matrix(y_test, y_pred_KNN)
print(cm_KNN ,'\n')

prediction_KNN = accuracy_score(y_test, y_pred_KNN)
print('Accuracy is :', prediction_KNN * 100)
```

```
[[11  0  0]
 [ 0 13  0]
 [ 0  1  5]]

Accuracy is : 96.66666666666667
```

In [27]:
```
plot_confusion_matrix(classifier_KNN, X_test, y_test)
plt.show()
```



5. KNN model and apply PAC

Our KNN model gave ~97% accuracy with approx error rate ~0.14,

m >= [1/e(ln(|H|) + ln(1/delta))]

|H| = 761530 (352343*22) #uinque number of data in each of the 4 cols

delta = 0.03

e <= 0.14

Our Target labels are: Iris-setosa, Iris-versicolor, Iris-virginica

In [39]:
```
import math

m = (math.log(761530, 2.718) + math.log(1/0.03, 2.718)) / 0.14
print('The number of samples required for the accuracy to be ~97% is : ', m )
```

```
The number of samples required for the accuracy to be ~97% is :  121.79579076342789
```

6. RandomForest

In [28]:
```
from sklearn.ensemble import RandomForestClassifier

classifier_RF = RandomForestClassifier(n_estimators = 50,
                                       random_state = 0)
classifier_RF.fit(X_train, y_train)
y_pred_RF = classifier_RF.predict(X_test)
cm_RF = confusion_matrix(y_test, y_pred_RF)
print(cm_RF ,'\n')
prediction_RF = accuracy_score(y_test, y_pred_RF)
print('Accuracy is :', prediction_RF * 100)
```

```
[[11  0  0]
 [ 0 13  0]
 [ 0  1  5]]

Accuracy is : 96.66666666666667
```

In [29]:
```
classifier_RF = RandomForestClassifier(n_estimators = 100,
```

```
                                        random_state = 0)
classifier_RF.fit(X_train, y_train)
y_pred_RF = classifier_RF.predict(X_test)
cm_RF = confusion_matrix(y_test, y_pred_RF)
print(cm_RF ,'\n')
prediction_RF = accuracy_score(y_test, y_pred_RF)
print('Accuracy is :', prediction_RF * 100)
```

```
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]

Accuracy is : 100.0
```

In [31]:

```
classifier_RF = RandomForestClassifier(n_estimators = 150,
                                        random_state = 0)
classifier_RF.fit(X_train, y_train)
y_pred_RF = classifier_RF.predict(X_test)
cm_RF = confusion_matrix(y_test, y_pred_RF)
print(cm_RF ,'\n')
prediction_RF = accuracy_score(y_test, y_pred_RF)
print('Accuracy is :', prediction_RF * 100)
```

```
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]

Accuracy is : 100.0
```

7.Ada Boost Model

In [32]:

```
from sklearn.ensemble import AdaBoostClassifier
cl = AdaBoostClassifier(base_estimator = classifier_RF,
                        n_estimators = 50,
                        random_state = 0)
cl.fit(X_train, y_train)
y_pred_AB = cl.predict(X_test)
cm_AB = confusion_matrix(y_test, y_pred_AB)
print(cm_AB ,'\n')
prediction_AB = accuracy_score(y_test, y_pred_AB)
print('Accuracy is :', prediction_AB * 100)
```

```
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]

Accuracy is : 100.0
```

In [33]:

```
cl = AdaBoostClassifier(base_estimator = classifier_NB,
                        n_estimators = 30,
                        random_state = 0)
cl.fit(X_train, y_train)
y_pred_AB = cl.predict(X_test)
cm_AB = confusion_matrix(y_test, y_pred_AB)
print(cm_AB ,'\n')
prediction_AB = accuracy_score(y_test, y_pred_AB)
print('Accuracy is :', prediction_AB * 100)
```

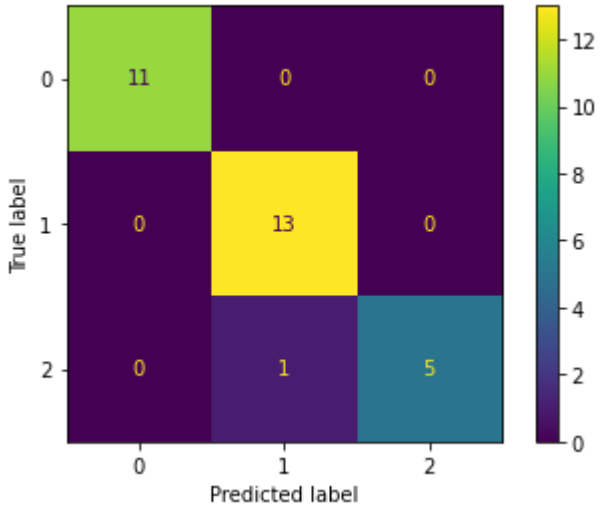
```
[[11  0  0]
 [ 0 12  1]
```

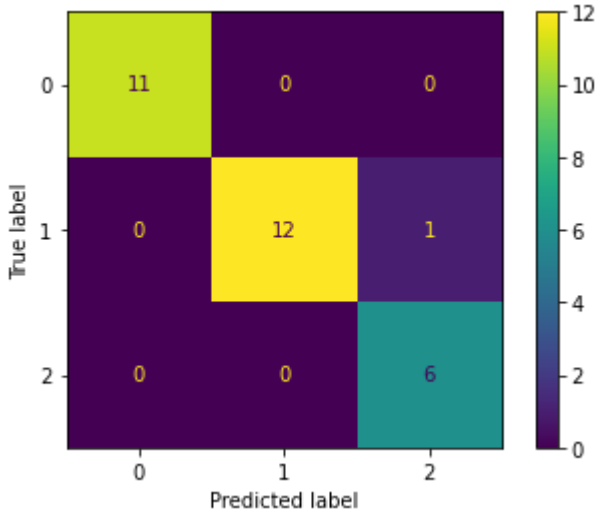
```
 [ 0  0  6]]

Accuracy is : 96.66666666666667
```

In [34]:

```
plot_confusion_matrix(classifier_NB, X_test, y_test)
plot_confusion_matrix(cl, X_test, y_test)
plt.show()
# Confusion matrix of Naive Bayes before AdaBoost and after ensembling with AdaBoost
```





Comaprision of all the classifiers implemented above:

The DecisionTree Classifier gave correct classification making the accuracy 100% for both 'Entropy' and 'Gini' criterions, Then the Naive Bayes Classiffier was implemented which gave 96.67 % accuracy with 1 missclassification in the 2nd label, Then Logistic Regression gave correct classification of all the data points. KNN-Classifier was implemented with N value of 3 which classified our data with ~97% accuracy. Then NaiveBayes classifier which had ~97% accuracy was taken to find out the 'm' value i.e, number of samples required for the classifier to predict or give accuracy of ~97%. RandomForest classifier was implemented with 3 different values of n_estimators = (50, 100, 150) which gave accuracies = (96.66, 100, 100) respectively, which implies as the number of n_estimators was increased the model gave better classification output. AdaBoost method of ensembling was applied to RandomForest classifier with n_estimator = 50 with corresponding accuracy ~97%, after the application of AdaBoost method the classification output was increased to 100% accuracy.

In [8]:
```
import pandas as pd
l1 = ['DecisionTree (entropy and gini)', 'Naive Bayes', 'Logistic Regression', 'K-Neare
l2 = [100, 96.67, 100, 100, 122, (50, 100, 150), 100]
```

In [16]:
```
tab = pd.DataFrame(list(zip(l1,l2)), columns = ['Classifier Model', 'Accuracy in Percen
tab
```

Out[16]:

	Classifier Model	Accuracy in Percentage
0	DecisionTree (entropy and gini)	100
1	Naive Bayes	96.67
2	Logistic Regression	100
3	K-Nearest Neighbor	100
4	Finding "m" values using PAC	122
5	RandomForest(n_estimators = 50, 100, 150)	(50, 100, 150)
6	AdaBoost	100

In [ ]:
```

```

Develop a linear regression using house price prediction dataset from UCI repository

In [76]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [77]:
```
dataset = pd.read_csv('USA_Housing.csv')
dataset.head()
```

Out[77]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

In [78]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Avg. Area Income             5000 non-null   float64
 1   Avg. Area House Age          5000 non-null   float64
 2   Avg. Area Number of Rooms    5000 non-null   float64
 3   Avg. Area Number of Bedrooms 5000 non-null   float64
 4   Area Population              5000 non-null   float64
 5   Price                        5000 non-null   float64
 6   Address                      5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

In [79]:
```
X = dataset.iloc[:, :5]
y = dataset.iloc[:, -2]
```

In [80]:
```
y.head()
```

Out[80]:
```
0    1.059034e+06
1    1.505891e+06
2    1.058988e+06
3    1.260617e+06
4    6.309435e+05
Name: Price, dtype: float64
```

In [81]:
```
# Test Train Split
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, random_state
```

In [82]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[82]:
```
LinearRegression()
```

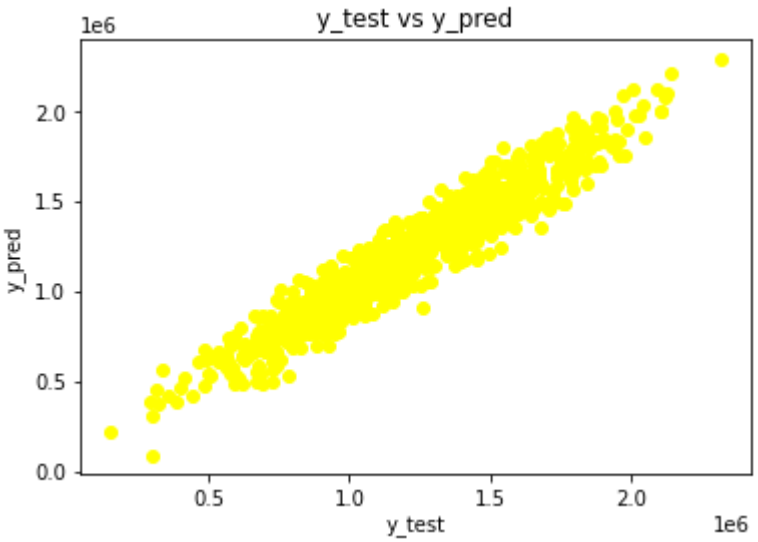
In [83]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
regressor.score(X_test, y_test).round(3)*100
```

Out[83]:
```
91.5
```

In [84]:
```
plt.title('y_test vs y_pred')
plt.xlabel('y_test')
plt.ylabel('y_pred')

plt.scatter(y_test, y_pred, c = 'yellow' )
```

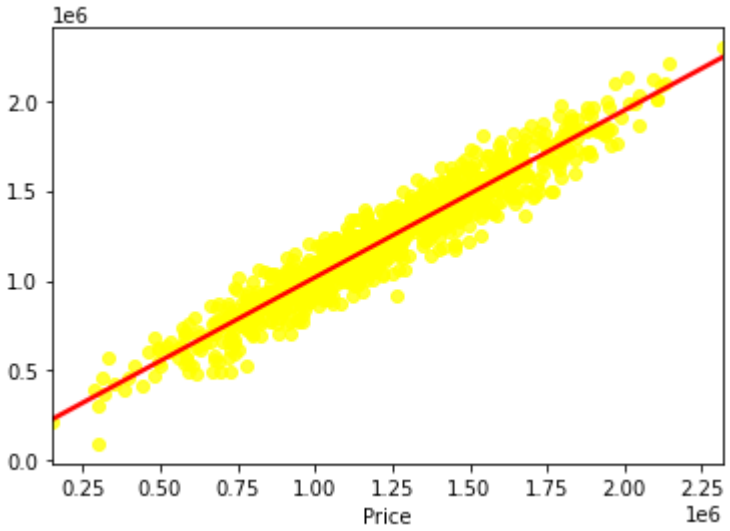
Out[84]:
```
<matplotlib.collections.PathCollection at 0x1445f5d1070>
```



In [85]:
```
sns.regplot(y_test, y_pred, scatter_kws = {'color' : 'yellow'}, line_kws = {'color':'re
```

```
C:\Users\Vishnu Prabhas\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWar
```

```
ning: Pass the following variables as keyword args: x, y. From version 0.12, the only va
lid positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```



1.Write the Python code to compute entropy and information gain

2. Write the Python code to demonstrate conditional probability

3. Write the Python code to compute Euclidean Distance between data points

4. Write the Python code to calculate covariance matrix, Eigen values and Eigen vectors

5. Write the Python code to calculate the following

```
Accuracy e
Misclassification
Type-1 and Type-2 error rates
Sensitivity
Specificity
```

1.Write the Python code to compute entropy and information gain

ENTROPY: Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

Where, p+ is the proportion of positive examples in S p– is the proportion of negative examples in S.

INFORMATION GAIN:

Information gain, is the expected reduction in entropy caused by partitioning the examples according to this attribute. The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

In [2]:

```
import numpy as np
import pandas as pd
import math
```

In [6]:

```
data = pd.read_csv('dataset.csv')
#data.head()
target = data.iloc[:,-1]
```

In [9]:

```
def entropy(target):
    elements, counts = np.unique(target, return_counts = True)
    entropy = np.sum([(-counts[i] / np.sum(counts)) * np.log2(counts[i] / np.sum(counts)
)
                      for i in range(len(elements))])
    return entropy
```

In [11]:

```
ent = entropy(target)
ent
```

Out[11]:

```
0.9402859586706311
```

In [17]:

```
def info_Gain(data, split_attr, target_name = 'answer'):
    total_entropy = entropy(data[target_name])
    vals, counts = np.unique(data[split_attr], return_counts = True)
    weighted_entropy = np.sum([(counts[i] / np.sum(counts)) * entropy(data.where(data[sp
lit_attr] == vals
                                                                                  [i]).d
ropna()
                                                                [target_name]) fo
r i in
                                                                 range(len(vals))
])
    InformationGain = total_entropy - weighted_entropy
    return InformationGain
```

In [19]:

```
info_outlook = info_Gain(data, "outlook", "answer")
info_temperature = info_Gain(data, "temperature", "answer")
info_humidity = info_Gain(data, "humidity", "answer")
info_wind = info_Gain(data, "wind", "answer")
```

In [22]:

```
print("Information Gain of Outlook: {}
      \n Information Gain of Temperature: {} \n Information Gain of Humidity: {}
      \n Information Gain of Wind: {}".format(info_outlook, info_temperature, info_humid
ity, info_wind))
```

```
Information Gain of Outlook: 0.24674981977443933
 Information Gain of Temperature: 0.02922256565895487
 Information Gain of Humidity: 0.15183550136234159
 Information Gain of Wind: 0.04812703040826949
```

---

2. Write the Python code to demonstrate conditional probability

$$Entropy\,(S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

Prob(type=coupe|rating=A)

Prob(type=sedan|rating=A)

Prob(type=coupe|rating=B)

Prob(type=sedan|rating=B)

Prob(type=coupe|rating=C)

Prob(type=sedan|rating=C)

In [34]:

```
collection = {"company": ["ford", "chevy", "ford", "ford", "ford", "toyota"],
              "model": ["mustang", "camaro", "fiesta", "focus", "taurus", "camry"],
              "rating": ["A", "B", "C", "A", "B", "B"],
              "type": ["coupe", "coupe", "sedan", "sedan", "sedan", "sedan"]}
df = pd.DataFrame(collection)
df
```

Out[34]:

	company	model	rating	type
0	ford	mustang	A	coupe
1	chevy	camaro	B	coupe
2	ford	fiesta	C	sedan
3	ford	focus	A	sedan
4	ford	taurus	B	sedan
5	toyota	camry	B	sedan

In [37]:

```
df_s = df.groupby('rating')['type'].value_counts() / df.groupby('rating')['type'].count(
)
df_f = df_s.reset_index(name ='conditional_probability')
df_f #conditional_prababilities
```

Out[37]:

	rating	type	conditional_probability
0	A	coupe	0.500000
1	A	sedan	0.500000
2	B	sedan	0.666667
3	B	coupe	0.333333
4	C	sedan	1.000000

---

3. Write the Python code to compute Euclidean Distance between data points

In [26]:

```
import math
def calculateDistance(x1, y1, x2, y2):
    dist = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
    return dist
```

In [41]:

```
print('Euclidean Distance between the given points is: ', calculateDistance(7, 7, 3, 7))
print('Euclidean Distance between the given points is: ', calculateDistance(3, 4, 3, 7))
print('Euclidean Distance between the given points is: ', calculateDistance(1, 4, 3, 7))
```

```
Euclidean Distance between the given points is:  4.0
Euclidean Distance between the given points is:  3.0
Euclidean Distance between the given points is:  3.605551275463989
```

---

4. Write the Python code to calculate covariance matrix, Eigen values and Eigen vectors

In [42]:

```
x = np.array([9, 15, 25, 14, 10])
y = np.array([39, 56, 93, 61, 50])
z = np.array([x, y])
print(x, y)
```

```
[ 9 15 25 14 10] [39 56 93 61 50]
```

In [43]:

```
cov = np.cov(x, y)
print('Covariance is :\n', cov)
```

```
Covariance is :
 [[ 40.3  126.15]
 [126.15 411.7 ]]
```

In [45]:

```
import scipy.linalg as la
eigen_vals, eigen_vectors = la.eig(cov)
print('The Eigen values are :\n', eigen_vals.real, "\n")
print('The Eigen vectors are :\n', eigen_vectors.real, "\n")
```

```
The Eigen values are :
 [  1.50431519 450.49568481]

The Eigen vectors are :
 [[-0.95582095 -0.29394949]
 [ 0.29394949 -0.95582095]]
```

---

5. Write the Python code to calculate the following

Accuracy e
Misclassification
Type-1 and Type-2 error rates
Sensitivity
Specificity

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

In [64]:

```
# Accuracy = (TP + TN)/(TP + TN + FP + FN)
# where: TP = True positive; FP = False positive; TN = True negative; FN = False negative
# consider the following values;
TP = 42
TN = 32
FP = 8
FN = 18

Accuracy = (TP + TN)/(TP + TN + FP + FN)
print('The Accuracy is {} %'.format(Accuracy*100))
```

```
The Accuracy is 74.0 %
```

In [65]:

```
# Misclassification = (FP + FN)/(TP + TN + FP + FN)

Misclassification = (FP + FN)/(TP + TN + FP + FN)
print('The Misclassification is {} %'.format(Misclassification*100))
```

```
The Misclassification is 26.0 %
```

Type 1 error is our FalsePositive value which is (FP) = 8

Type 1 error is our FalsePositive value which is (FP) = 8

Type 2 error is our FalseNegative value which is (FN) = 18

In [67]:

```
# Sensitivity = (TP) / (TP + FN)
Sensitivity = (TP) / (TP + FN)
print('The Sensitivity is {} %'.format(Sensitivity*100))
```

```
The Sensitivity is 70.0 %
```

In [68]:

```
# Specificity = (TN) / (TN + FP)
Specificity = (TN) / (TN + FP)
print('The Specificity is {} %'.format(Specificity*100))
```

```
The Specificity is 80.0 %
```

Name: VIVEK V KINI

Registration Number: 201046036

Big Data And Data Analytics

FML LAB ASSIGNMENT-1

FML LAB ASSIGNMENT 1

Q1. Write a program to prompt the user for hours and rate per hour to compute gross pay.Hour = 35 & Rate = 2.75

In [2]:

```
# To take integer input from user
hour=int(input('Enter Hours: '))
x=hour
rate_per_hour=float(input('Enter Rate per Hour: '))

Gross_pay=hour*rate_per_hour
print("Gross pay = {} Rs/-".format(Gross_pay))
```

```
Enter Hours: 96
Enter Rate per Hour: 13.5
Gross pay = 1296.0 Rs/-
```

Q2. Which will never be printed in the following two codes sets?

In [ ]:

```
# code snippet to test
if x<2:
  print('below 2')
elif x>=20:
  print('Two or more')
else:
  print('something else')
```

Ans : In this code snippet 'Something else' will never gets printed

In [ ]:

```
# Code snippet to test
if x<2:
  print('Below 2')
elif x<20:
  print('Below 20')
elif x<10:
  print('Below 10')
else:
  print('Something else')
```

Ans: In this code snippet Below 10 will never gets printed

Q3.Rewrite the program-1using try and except to prompt the user for hours and rate per hour to compute gross pay.(if non-numeric inputsentered, it should except).Hour = 35 & Rate = 2.75.

In [ ]:

```
# Q1 using try and except. try will run the code in its block and if any non-numeric valu
e is enterd by user it will throw the control to except and display the error message
try:
  hour=int(input('Enter Hours: '))
  rate_per_hour=float(input('Enter Rate per Hour: '))
except:
  print("Please enter numeric values")
else:
  Gross_pay=hour*rate_per_hour
  print("Gross pay = {} Rs/-".format(Gross_pay))
```

```
Enter Hours: 35
Enter Rate per Hour: 2.75
Gross pay = 96.25 Rs/-
```

Q4. Rewrite the program-1with time-and-a half for overtime and create a function called paycomp which takes two parameters (hoursand rate per hour).Hours = 45 and rate = 10.

In [ ]:

```
hour=int(input('Enter Hours: '))
rate_per_hour=float(input('Enter Rate per Hour: '))
overtime=1.5

hour=hour+overtime

Gross_pay=hour*rate_per_hour
print("Gross pay = {} Rs/-".format(Gross_pay))
```

```
Enter Hours: 45
Enter Rate per Hour: 10
Gross pay = 465.0 Rs/-
```

Q5. What is this code doing?

In [ ]:

```
# Infinite running loop
n=5
while n>0:
  print('Lather')
  print('Rinse')
print('Dry off!')
```

Ans: This code is running for infinite number of times. It is an infinite loop and keeps on printing Lather Rinse

Q6.What is this code doing?

In [ ]:

```
n=0
while n>0:
  print('Lather')
  print('Rinse')
print('Dry off!')
```

Ans: This code will just print dry off because looping condition is false so it is not entering in the while loop

Q7.Consider the list of elements [9, 41, 23, 54, 33, 21, 8] use for loop to find

a.Largest number

b.Smallestnumber

c.Number of numbers

d.Number of odd number

e.Number of even numbers

f.Number of prime numbers

g.Sum and average of numbers

h.Filter the numbers greater than 20

i.Filter the numbers lessthan 15

j.Search for the number 3

In [ ]:

```
List=[9,41,23,54,33,21,8]
print(List)
```

```
[9, 41, 23, 54, 33, 21, 8]
```

In [ ]:

```
# a) LARGEST NUMBER

large=-1

for iterate in List:
  if iterate>large:
    large=iterate

print("Largest number is: {}".format(large))
```

```
Largest number is: 54
```

In [ ]:

```
# b) SMALLEST NUMBER

small=List[0]

for iterate in List:
  if iterate<small:
    small=iterate

print("Smallest number is: {}".format(small))
```

```
Smallest number is: 8
```

In [ ]:

```
# c) Number of Numbers
count=0
for iterate in List:
  count+=1
print("Total number of numbers in list: {}".format(count))
```

```
Total number of numbers in list: 7
```

In [ ]:

```
# d) Number of odd number
count=0
Oddlist=[]
for iterate in List:
  if((iterate%2)!=0):
    count+=1
    Oddlist.append(iterate)
print("Total number of odd number: {} i.e {}".format(count,Oddlist))
```

```
Total number of odd number: 5 i.e [9, 41, 23, 33, 21]
```

In [ ]:

```
# e) Number of even number
count=0
evenlist=[]
for iterate in List:
  if((iterate%2)==0):
    count+=1
    evenlist.append(iterate)
print("Total number of even number: {} i.e {}".format(count,evenlist))
```

```
Total number of even number: 2 i.e [54, 8]
```

In [ ]:

```
# f) Number of prime numbers
count=0
Primelist=[]
for num in List:
        if num <= 1:
            continue
        for iterate in range(2, num):
            if (num % iterate) == 0:
                break
        else:
            count+=1
            Primelist.append(num)

print("Number of prime number : {} i.e {}".format(count,Primelist))
```

```
Number of prime number : 2 i.e [41, 23]
```

In [ ]:

```
# g) Sum and average of numbers
sum=0
count=0
for num in List:
  sum+=num
  count+=1

average=sum/count

print('Sum and average of the numbers : {} and {}'.format(sum,average))
```

```
Sum and average of the numbers : 189 and 27.0
```

In [ ]:

```
# h) Filter number greater than 20
num=20
new_list=[]
for number in List:
  if number>num:
    new_list.append(number)

print('Numbers greater than 20: {}'.format(new_list))
```

```
Numbers greater than 20: [41, 23, 54, 33, 21]
```

In [ ]:

```
# i) Filter number less than 15
num=15
new_list=[]
for number in List:
  if number<num:
    new_list.append(number)

print('Numbers less than 15: {}'.format(new_list))
```

```
Numbers less than 15: [9, 8]
```

In [ ]:

```
# j) Search for number 3
number=3
found=False
for iterate in range(len(List)):
  if (List[iterate]==number):
    found=True
    print('Number exist at {} '.format(i))

if (found==False):
   print('Number not found in the list')
```

```
Number not found in the list
```

Q8.Illustrate the use of type operator and type conversion (use your own examples)

In [ ]:

```
#consider this code snippet
hour=input('enter a number') # Getting values from user using input function
rate=input('enter a number')
print('Type of num1= {} and num2= {}'.format(type(hour),type(rate)))  # printing the type of data that is stored in inputdata
gross_pay=hour*rate          # error we can't multiply 2 strings
print('Gross pay of this person is {}'.format(gross_pay))
```

```
enter a number2
enter a number3
Type of num1= <class 'str'> and num2= <class 'str'>

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-ef85530dee60> in <module>()
      3 rate=input('enter a number')
      4 print('Type of num1= {} and num2= {}'.format(type(hour),type(rate)))  # printing the type of data that is stored in inputdata
----> 5 gross_pay=hour*rate
      6 print('Gross pay of this person is {}'.format(gross_pay))
      7

TypeError: can't multiply sequence by non-int of type 'str'
```

So this code wants to calculate gross pay.

First ask for a user input that to be stored in hour(number of hours work) and rate (rate of work).

Now further we we want to multiply this 2 inputs to get our gross pay i.e gross_pay=hour*rate.

But when we run this code it throws error saying can't multiply sequence by non-int of type 'str'. This is because when we take user input using input function it stores the value with string class even if we give input as a number.

So to check what type of input we get we use type() function which gives class of hour and rate is str i.e string.

So here where type convertion takes place. we have to explicitly convert the type of str to integer as shown in the following code snippet.

Now after we convert the type of data using int() it converts the input data to integer if we want we can use float() also for rate input. This we can see using type() function.

In [ ]:

```
# Type convertion code snippet
hour=int(input('enter a number')) # Getting values from user using input function
rate=int(input('enter a number'))
gross_pay=hour*rate
print('Gross pay of this person is {}'.format(gross_pay))
```

```
print('Type of num1= {} and num2= {}'.format(type(hour),type(rate)))  # printing the type of data that is stored in inputdata
```

```
enter a number5
enter a number3
Gross pay of this person is 15
Type of num1= <class 'int'> and num2= <class 'int'>
```

Q9.Illustrate the use of break and continue with your own examples

Break

Ans. The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop. If the break statement is inside a nested loop (loop inside another loop), the break statement will terminate the innermost loop.

In [ ]:

```
string="MACHINELEARNING"
for val in string:
    if val == "I":
        break
    print(val)
```

```
M
A
C
H
```

In this code we are iterating over string that contains MACHINELEARNING and when I comes we break the loop and comes out of it

Continue

Ans. The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

In [ ]:

```
string="MACHINELEARNING"
for val in string:
    if val == "I":
        continue
    print(val)
```

```
M
A
C
H
N
E
L
E
A
R
N
N
G
```

In this code we iterate over string containing MACHINELEARNING and we skip over I character with the help of continue

Name: VIVEK V KINI

Registration Number: 201046036

Big Data And Data Analytics

FML LAB ASSIGNMENT-2

Q1.Store 1 and 2 provide price of items. Write a code to suggest which item can be prchased from store 1 and whcih items from store 2, using lists given below.

store1 = [10.00, 11.00, 12.34, 2.34]

store2 = [19.00, 0.10, 12.34, 2.01]

In [1]:

```
#creating 2 list for 2 stores having price of items in them
store1 = [10.00, 11.00, 12.34, 2.34]
store2 = [19.00, 0.10, 12.34, 2.01]



# now creating a function which returns which items to bought from where
def Purchase(store1,store2):
  length=len(store1) # since length of both stores is same stored length in a variable(c
hoose any one store)
  for iterate in range(length):
    if (store1[iterate]<store2[iterate]):
      print('Purchase item {} from store1 having price {} Rs/-'.format(iterate+1,store1[
iterate]))
    elif (store1[iterate]==store2[iterate]):
      print('Purchase item {} from any store1 or store2 both have price {} Rs/-'.format(
iterate+1,store1[iterate]))
    else:
      print('Purchase item {} from store2 having prince {} Rs/-'.format(iterate+1,store2
[iterate]))
```

In [2]:

```
Purchase(store1,store2) #calling function Purchase by passing store1 and store2 list as a
n argument
```

```
Purchase item 1 from store1 having price 10.0 Rs/-
Purchase item 2 from store2 having prince 0.1 Rs/-
Purchase item 3 from any store1 or store2 both have price 12.34 Rs/-
Purchase item 4 from store2 having prince 2.01 Rs/-
```

FML LAB PRACTICE ASSIGNMNET

Name: VIVEK V KINI

Registration Number: 201046036

Big Data And Data Analytics

Q1. LIST CHEAPEST PRICE OF MILK,WATER,SUGAR AND TEA PACKET USING MAP AND MIN FUNCTION

In [1]:

```
store1=[10.00,11.00,12.34,5.34]
store2=[19.00,10.10,12.34,5.01]
store3=[15.00,12.00,11.00,6.00]
Products=['MILK','WATER','SUGAR','TEA']
Cheapest=[item for item in map(min,store1,store2,store3)] #list comprehension
for items in range(len(Cheapest)):
  print("Buy {} Product from store{} at {}Rs\-".format(Products[items],items+1,Cheapest[
items]))
```

```
Buy MILK Product from store1 at 10.0Rs\-
Buy WATER Product from store2 at 10.1Rs\-
Buy SUGAR Product from store3 at 11.0Rs\-
Buy TEA Product from store4 at 5.01Rs\-
```

Q2. Prepare Multiplication table upto table 10

In [2]:

```
table=[j*i for i in range(1,11) for j in range(1,11)] #creating a list containg all the
multiplication values
table_update = [table[i:i + 10] for i in range(0, len(table), 10)] #updating the above l
ist to seperate them into different tables
for item in range(len(table_update)):
  print("TABLE OF {} IS {}".format(item+1,table_update[item]))
```

```
TABLE OF 1 IS [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
TABLE OF 2 IS [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
TABLE OF 3 IS [3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
TABLE OF 4 IS [4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
TABLE OF 5 IS [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
TABLE OF 6 IS [6, 12, 18, 24, 30, 36, 42, 48, 54, 60]
TABLE OF 7 IS [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
TABLE OF 8 IS [8, 16, 24, 32, 40, 48, 56, 64, 72, 80]
TABLE OF 9 IS [9, 18, 27, 36, 45, 54, 63, 72, 81, 90]
TABLE OF 10 IS [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Q3. Take an array from 0 to 99 and then convert that to 10 by 10 matrix then replace the 1st 3 rows and 3 columns by 0 and then replace last 3 rows and 1st 3 columns by -1

In [3]:

```
import numpy as np
```

In [4]:

```
array=np.arange(100)   # create an array containing 100 elements
array.resize(10,10)    #resizing it to 10x10 matrix
print(array)
```

```
[[ 0  1  2  3  4  5  6  7  8  9]
```

```
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
```

In [5]:

```
array[0:3,0:3]=0  #replace 1st 3rows and 3 columns to 0
```

In [6]:

```
array[-3:,0:3]=-1 #replace last 3 rows and 3 columns to -1
```

In [7]:

```
print(array)
```

```
[[ 0  0  0  3  4  5  6  7  8  9]
 [ 0  0  0 13 14 15 16 17 18 19]
 [ 0  0  0 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [-1 -1 -1 73 74 75 76 77 78 79]
 [-1 -1 -1 83 84 85 86 87 88 89]
 [-1 -1 -1 93 94 95 96 97 98 99]]
```

Name: VIVEK V KINI

Registration Number: 201046036

Big Data And Data Analytics

FML LAB ASSIGNMENT PANDAS

In [1]:

```
#LIBRARIES
import pandas as pd
import numpy as np
```

In [2]:

```
data=[1,2,3,4,5,6]
series=pd.Series(data,index=['a','b','c','d','e','f'])
#to access first 3 elements of data from indices
print('Series elements accessed by indices :')
print(series[0],series[1],series[2])
#to access last 3 elements of data from index values
print('Series elements accessed by index given:')
print(series['d'],series['e'],series['f'])
```

```
Series elements accessed by indices :
1 2 3
Series elements accessed by index given:
4 5 6
```

Q1.[[0 0 1]
[0 0 1]]

In [3]:

```
arr=np.array([[0,0,1],
              [0,0,1]])
print(arr)
```

```
[[0 0 1]
 [0 0 1]]
```

Q2.

index value

0 Tiger

1 Bear

2 Moose

In [4]:

```
data={'index':[0,1,2],'value':['Tiger','Bear','Moose']}
dataf=pd.DataFrame(data,columns=['index','value'])
print(dataf.to_string(index=False))
```

```
 index  value
     0  Tiger
     1   Bear
     2  Moose
```

Q3.

index value

0 1

1 2

2 3

In [5]:

```
data={'index':[0,1,2],'value':['1','2','3']}
dataf=pd.DataFrame(data,columns=['index','value'])
print(dataf.to_string(index=False))
```

```
 index value
     0     1
     1     2
     2     3
```

Q4.

index value

0 1.0

1 2.0

2 NaN

In [6]:

```
data={0:1.0,1:2.0}
S=pd.Series(data,index=[0,1,2])
print(S)
```

```
0    1.0
1    2.0
2    NaN
dtype: float64
```

Q4.

import numpy as np

np.nan == None

Ans:

numpy.nan is IEEE 754 floating point representation of Not a Number (NaN), which is of Python build-in numeric type float.

However, None is of NoneType and is an object.

In [7]:

```
print(type(np.nan))
print(type(None))
```

```
<class 'float'>
<class 'NoneType'>
```

So when we compare np.nan and None we get False

In [8]:

```
np.nan==None
```

Out[8]:

```
False
```

Q5.np.nan == np.nan

In [9]:

```
np.nan==np.nan
```

Out[9]:

```
False
```

Ans

For comparison purposes, np.nan compared to another np.nan using == returns False because we are comparing 2 different objects, while np.nan compared to another np.nan using is returns True.

Using both == and is, None compared to another None returns True.

In [10]:

```
np.nan is np.nan
```

Out[10]:

```
True
```

In [11]:

```
None==None
```

Out[11]:

```
True
```

In [12]:

```
None is None
```

Out[12]:

```
True
```

np.isnan(np.nan)

In [13]:

```
np.isnan(np.nan)
```

Out[13]:

```
True
```

Ans

np.isnan() Test element-wise for NaN and return result as a boolean array. so when we compare np.nan it returns True as a result

Q6.

Write the code to get this output

Cricket India

Golf Scotland

Sumo Japan

Taekwondo South Korea

dtype: object

In [14]:

```
data={'cricket':'India','golf':'Scotland','Sumo':'Japan','Taekwondo':'South Korea'}
S=pd.Series(data)
print(S)
```

```
cricket            India
golf            Scotland
Sumo               Japan
Taekwondo    South Korea
dtype: object
```

Q7.

Golf Scotland

Sumo Japan

Hockey NaN

dtype: object

In [15]:

```
data={'golf':'Scotland','Sumo':'Japan'}
S=pd.Series(data,index=['golf','Sumo','Hockey'])
print(S)
```

```
golf      Scotland
Sumo         Japan
Hockey         NaN
dtype: object
```

Q8.

To find the time taken for execution generate a big series of random numbers and calculate the time.

In [16]:

```
%timeit np.random.randint(0,1000,10000)
```

```
The slowest run took 11.16 times longer than the fastest. This could mean that an interme
diate result is being cached.
10000 loops, best of 3: 54.4 µs per loop
```

Q9.

Name Item Purchased Cost

Store1 Chris Milk 22.5

Store2 Kevyn Bread 12.5

Store3 Vinod Butter 15.0

In [17]:

```
data={'Name':['Chris','Kevyn','Vinod'],'Item Purchased':['Milk','Bread','Butter'],'Cost'
:[22.5,12.5,15.0]}
df=pd.DataFrame(data,index=['Store1','Store2','Store3'])
print(df)
```

```
         Name Item Purchased  Cost
Store1  Chris           Milk  22.5
Store2  Kevyn          Bread  12.5
Store3  Vinod         Butter  15.0
```

Q10.

print the cost of all the stores

add 12 to all elements of cost column

delete elements of column “Name”

In [18]:

```
data={'Name':['Chris','Kevyn','Vinod'],'Item Purchased':['Milk','Bread','Butter'],'Cost'
:[22.5,12.5,15.0]}
df=pd.DataFrame(data,index=['Store1','Store2','Store3'])
df
```

Out[18]:

	Name	Item Purchased	Cost
Store1	Chris	Milk	22.5
Store2	Kevyn	Bread	12.5
Store3	Vinod	Butter	15.0

In [19]:

```
print('AFTER ADDING 12 TO EACH COST ELEMENT:')
df['Cost']+=12
print(df)
print(" ")
print('AFTER DELETING NAME COLUMN:')
del df['Name']
print(df)
```

```
AFTER ADDING 12 TO EACH COST ELEMENT:
         Name Item Purchased  Cost
Store1  Chris           Milk  34.5
Store2  Kevyn          Bread  24.5
Store3  Vinod         Butter  27.0

AFTER DELETING NAME COLUMN:
       Item Purchased  Cost
Store1           Milk  34.5
Store2          Bread  24.5
Store3         Butter  27.0
```

Q.11

from the given code

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

Do the following

selected row ‘c’ by passing row label

select row ‘c’ by passing integer location

remove rows 2 and 3

In [20]:

```
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df=pd.DataFrame(d)
print(df)
```

```
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
```

In [21]:

```
# SELECT A ROW 'c' by passing row label
df.loc['c']
```

Out[21]:

```
one    3.0
two    3.0
Name: c, dtype: float64
```

In [22]:

```
#SELECT ROW 'c' by using interger location
df.iloc[2]
```

Out[22]:

```
one    3.0
two    3.0
Name: c, dtype: float64
```

In [23]:

```
print('After Deletion:')
data=df.drop(['c','d'])
print(data)
```

```
After Deletion:
   one  two
a  1.0    1
b  2.0    2
```