Course name: **Python programming and analytics by Rahul Sir**

Topic name: **decision tree part2**

Video name: **Sumit class python**

Video length: **24 minutes 39 seconds**

Python packages used

- **NumPy**
  - NumPy is a Numeric Python module. It provides fast mathematical functions.
  - NumPy provides robust data structures for efficient computation of multi-dimensional arrays & matrices.
  - We used NumPy to read data files into NumPy arrays and data manipulation.
- **Pandas**
  - Provides Data Frame Object for data manipulation
  - Provides reading & writing data b/w different files.
  - Data Frames can hold different types data of multidimensional arrays.
- **Scikit-Learn**
  - It's a machine learning library. It includes various machine learning algorithms.
  - We are using its
    - train_test_split,
    - DecisionTreeClassifier,

- accuracy_score algorithms.

Importing Required Libraries and data

Let's first load the required libraries and the data

```
In [41]: import pandas as pd
```

```
In [42]: data = pd.read_csv("C:/Users/Sahibjot/Desktop/Data_set_telecom_churn.csv")
```

```
In [43]: data.describe()
```

Out[43]:

	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Int'l Plan	VMail Plan	Day Calls	Day Charge	Ev
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.
mean	101.064806	8.099010	179.775098	200.980348	200.872037	10.237294	1.562856	0.096910	0.276628	100.435644	30.562307	100
std	39.822106	13.688365	54.467389	50.713844	50.573847	2.791840	1.315491	0.295879	0.447398	20.069084	9.259435	19.
min	1.000000	0.000000	0.000000	0.000000	23.200000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	74.000000	0.000000	143.700000	166.600000	167.000000	8.500000	1.000000	0.000000	0.000000	87.000000	24.430000	87.
50%	101.000000	0.000000	179.400000	201.400000	201.200000	10.300000	1.000000	0.000000	0.000000	101.000000	30.500000	100.
75%	127.000000	20.000000	216.400000	235.300000	235.300000	12.100000	2.000000	0.000000	1.000000	114.000000	36.790000	114.
max	243.000000	51.000000	350.800000	363.700000	395.000000	20.000000	9.000000	1.000000	1.000000	165.000000	59.640000	170.

```
In [44]: data.shape
```

Out[44]: (3333, 21)

```
In [45]: data.dtypes

Out[45]: Phone              object
         Account Length      int64
         VMail Message       int64
         Day Mins          float64
         Eve Mins          float64
         Night Mins        float64
         Intl Mins         float64
         CustServ Calls      int64
         Int'l Plan          int64
         VMail Plan          int64
         Day Calls           int64
         Day Charge        float64
         Eve Calls           int64
         Eve Charge        float64
         Night Calls         int64
         Night Charge      float64
         Intl Calls          int64
         Intl Charge       float64
         State              object
         AreaCode            int64
         Churn               int64
         dtype: object
```

to see some of the core statistics about a particular column, you can use the 'describe' function.

- For numeric columns, describe() returns basic statistics: the value count, mean, standard deviation, minimum, maximum, and 25th, 50th, and 75th quantiles for the data in a column.
- For string columns, describe() returns the value count, the number of unique entries, the most frequently occurring value ('top'), and the number of times the top value occurs ('freq')

Select a column to describe using a string inside the [] braces, and call describe()

Many DataFrames have mixed data types, that is, some columns are numbers, some are strings, and some are dates etc. Internally, CSV files do not contain information on what data types are contained in each column; all of the data is just characters. Pandas infers the data types when loading the data, e.g. if a column contains only numbers, pandas will set that column's data type to numeric: integer or float.

You can check the types of each column in our example with the '.dtypes' property of the dataframe.

```
AreaCode              int64
Churn                 int64
dtype: object
```

```
data['AreaCode']= data.AreaCode.astype(str)
#data['Churn']= data.Churn.astype(str)
```

Now, as you can see the area code column is denoted with integer data type, but the area code can't be an integer, it must be a character or string as you can't take the average of area code, so we need to change this integer column into string column.

In some cases, the automated inferring of data types can give unexpected results. Note that strings are loaded as 'object' datatypes.

To change the datatype of a specific column, use the .astype() function

```
In [47]: data.isnull().any()

Out[47]: Phone                False
         Account Length       False
         VMail Message        False
         Day Mins             False
         Eve Mins             False
         Night Mins           False
         Intl Mins            False
         CustServ Calls       False
         Int'l Plan           False
         VMail Plan           False
         Day Calls            False
         Day Charge           False
         Eve Calls            False
         Eve Charge           False
         Night Calls          False
         Night Charge         False
         Intl Calls           False
         Intl Charge          False
         State                False
         AreaCode             False
         Churn                False
         dtype: bool

In [48]: data.isnull().values.sum()

Out[48]: 0

In [49]: data.isnull().values.any()

Out[49]: False
```

Pandas **Series.isnull()** function detect missing values in the given series object. It returns a boolean same-sized object indicating if the values are NA. Missing values gets mapped to True and non-missing value gets mapped to False.

As we can see in the output, the Series.isnull() function has returned an object containing boolean values. All values have been mapped to False because there is no missing value in the given series object.

Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning. Label encoding convert the data in machine readable form, but it assigns a unique number(starting from 0) to each class of data.

```
In [53]: from sklearn.preprocessing import LabelEncoder
         lb_make = LabelEncoder()
         data["Area_code"] = lb_make.fit_transform(data["AreaCode"])
         data["State_code"] = lb_make.fit_transform(data["State"])
```

```
In [54]: data.shape

Out[54]: (3333, 23)
```

```
In [55]: data.head()
```

Out[55]:

	Phone	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Int'l Plan	VMail Plan	...	Eve Charge	Night Calls	Night Charge	Intl Calls	Intl Charge	State	AreaCode	Churn	Area_coc
0	382-4657	128	25	265.1	197.4	244.7	10.0	1	0	1	...	16.78	91	11.01	3	2.70	KS	415	0	
1	371-7191	107	26	161.6	195.5	254.4	13.7	1	0	1	...	16.62	103	11.45	3	3.70	OH	415	0	
2	358-1921	137	0	243.4	121.2	162.6	12.2	0	0	0	...	10.30	104	7.32	5	3.29	NJ	415	0	
3	375-9999	84	0	299.4	61.9	196.9	6.6	2	1	0	...	5.26	89	8.86	7	1.78	OH	408	0	
4	330-6626	75	0	166.7	148.3	186.9	10.1	3	1	0	...	12.61	121	8.41	3	2.73	OK	415	0	

5 rows × 23 columns

Now, the next step is defining X and Y variables:

```
In [58]: cols = ['Account Length', 'VMail Message', 'Day Mins', 'Eve Mins', 'Night Mins', 'Intl Mins', 'CustServ Calls', "Int'l Plan", 'Vl
```

```
In [59]: x=data[cols]
         y=data['Churn']
```

Now, the next step is to split the data:

```
In [62]: from sklearn import tree
         from sklearn.tree import DecisionTreeClassifier
         from sklearn import metrics
         from sklearn.model_selection import train_test_split
```

```
In [63]: #Decision Tree model

         X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

To understand model performance, dividing the dataset into a training set and a test set is a good strategy. Let's split dataset by using function train_test_split().

Machine learning is about learning some properties of a data set and then testing those properties against another data set. A common practice in machine learning is to evaluate an algorithm by splitting a data set into two. We call one of those sets the **training set**, on which we learn some properties; we call the other set the **testing set**, on which we test the learned properties.

You can't possibly manually split the dataset into two. And you also have to make sure you split the data in a random manner. To help us with this task, the SciKit library provides a tool, called the Model Selection library. There's a class in the library which is, aptly, named 'train_test_split.' Using this we can easily split the dataset into the training and the testing datasets in various proportions.

- **test_size** — This parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.3 as the value, the dataset will be split 30% as the test dataset.
- **train_size** — You have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.
- **random_state** — Here you pass an integer, which will act as the seed for the random number generator during the split.

After this fit your model on the train set using fit()

Once the data has been divided into the training and testing sets, the final step is to train the decision tree algorithm on this data and make predictions. Scikit-Learn contains the tree library, which contains built-in classes/methods for various decision tree algorithms. Since we are going to perform a classification task here, we will use the DecisionTreeClassifier class for this example. The fit method of this class is called to train the algorithm on the training data, which is passed as parameter to the fit method

```
In [64]: DTC = DecisionTreeClassifier()
         DTC.fit(X_train, y_train)

Out[64]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                     splitter='best')
```

DecisionTreeClassifier(): This is the classifier function for DecisionTree. It is the main function for implementing the algorithms. Some important parameters are:

- **criterion:** It defines the function to measure the quality of a split. Sklearn supports "gini" criteria for Gini Index & "entropy" for Information Gain. By default, it takes "gini" value.
- **splitter:** It defines the strategy to choose the split at each node. Supports "best" value to choose the best split & "random" to choose the best random split. By default, it takes "best" value.
- **max_features:** It defines the no. of features to consider when looking for the best split. We can input integer, float, string & None value.
  - If an integer is inputted then it considers that value as max features at each split.
  - If float value is taken then it shows the percentage of features at each split.
  - If "auto" or "sqrt" is taken then max_features=sqrt(n_features).
  - If "log2" is taken then max_features= log2(n_features).
  - If None, then max_features=n_features. By default, it takes "None" value.
- **max_depth:** The max_depth parameter denotes maximum depth of the tree. It can take any integer value or None. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. By default, it takes "None" value.
- **min_samples_split:** This tells above the minimum no. of samples reqd. to split an internal node. If an integer value is taken then consider min_samples_split as the minimum no. If float, then it shows percentage. By default, it takes "2" value.

- **min_samples_leaf:** The minimum number of samples required to be at a leaf node. If an integer value is taken then consider min_samples_leaf as the minimum no. If float, then it shows percentage. By default, it takes "1" value.
- **max_leaf_nodes:** It defines the maximum number of possible leaf nodes. If None then it takes an unlimited number of leaf nodes. By default, it takes "None" value.
- **min_impurity_split:** It defines the threshold for early stopping tree growth. A node will split if its impurity is above the threshold otherwise it is a leaf.

Generally, importance provides a score that indicates how useful or valuable each feature was in the construction of the boosted decision trees within the model. The more an attribute is used to make key decisions with decision trees, the higher its relative importance.

This importance is calculated explicitly for each attribute in the dataset, allowing attributes to be ranked and compared to each other.

Importance is calculated for a single decision tree by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for. The performance measure may be the purity (Gini index) used to select the split points or another more specific error function.

The feature importances are then averaged across all of the the decision trees within the model.

```
In [83]: DTC.feature_importances_

Out[83]: array([0.02015418, 0.00533569, 0.08942887, 0.07988739, 0.01900483,
                0.07094019, 0.11734732, 0.07396806, 0.06994604, 0.01315782,
                0.17226219, 0.02191853, 0.07025791, 0.02446923, 0.0419426 ,
                0.06407253, 0.0204763 , 0.        , 0.02543031])
```

Now, perform prediction on the test set using predict()

```
In [65]: Preds= DTC.predict(X_test)
```

To check how much this model is accurate:

The function accuracy_score() will be used to print accuracy of Decision Tree algorithm. By accuracy, we mean the ratio of the correctly predicted data points to all the predicted data points. Accuracy as a metric helps to understand the effectiveness of our algorithm. It takes 4 parameters.

- y_true,
- y_pred,
- normalize,
- sample_weight.

Out of these 4, normalize & sample_weight are optional parameters. The parameter y_true accepts an array of correct labels and y_pred takes an array of predicted labels that are returned by the classifier. It returns accuracy as a float value.

```
In [68]: from sklearn.metrics import accuracy_score
         print('DT accuracy: {:.3f}'.format(accuracy_score(y_test, DTC.predict(X_test))))

DT accuracy: 0.900
```

Model Evaluation using Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model .In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix.

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm.
It allows easy identification of confusion between classes e.g. one class is commonly mislabeled as the other. Most performance measures are computed from the confusion matrix.

- Positive (P) : Observation is positive (for example: is an apple).
- Negative (N) : Observation is not positive (for example: is not an apple).
- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

At this point we have trained our algorithm and made some predictions. Now we'll see how accurate our algorithm is. For classification tasks some commonly used metrics are confusion matrix, precision, recall, and F1 score. Lucky for us Scikit=-Learn's metrics library contains the classification_report and confusion_matrix methods that can be used to calculate these metrics for us:

```
In [69]:
from sklearn.metrics import classification_report
print(classification_report(y_test, DTC.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94       862
           1       0.61      0.76      0.68       138

   micro avg       0.90      0.90      0.90      1000
   macro avg       0.79      0.84      0.81      1000
weighted avg       0.91      0.90      0.90      1000
```

In [70]:
```
pred = DTC.predict(X_test)
from sklearn.metrics import confusion_matrix
import seaborn as sns
forest_cm = metrics.confusion_matrix(pred, y_test)
sns.heatmap(forest_cm, annot=True, fmt='.2f',xticklabels = ["Churn", "Non Churn"] , yticklabels = ["Churn", "Non Churn"] )
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Decision Tree Classification')
```

Out[70]:
```
Text(0.5, 1.0, 'Decision Tree Classification')
```