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

Topic name: **decision tree classification**

Video name: **decision tree classification Sumit batch**

Video length: **1 hour 10 minutes 51 seconds**

As a marketing manager, you want a set of customers who are most likely to purchase your product. This is how you can save your marketing budget by finding your audience. As a loan manager, you need to identify risky loan applications to achieve a lower loan default rate. This process of classifying customers into a group of potential and non-potential customers or safe or risky loan applications is known as a classification problem. Classification is a two-step process, learning step and prediction step. In the learning step, the model is developed based on given training data. In the prediction step, the model is used to predict the response for given data. Decision Tree is one of the easiest and popular classification algorithms to understand and interpret. It can be utilized for both classification and regression kind of problem.

What is a Decision Tree? How does it work?

Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

Decision Tree is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

A **decision tree** is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

A decision tree is a flowchart-like structure in which each internal node represents a “test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of problem at hand (classification or regression). Decision Tree algorithms are referred to as **CART (Classification and Regression Trees)**.

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.

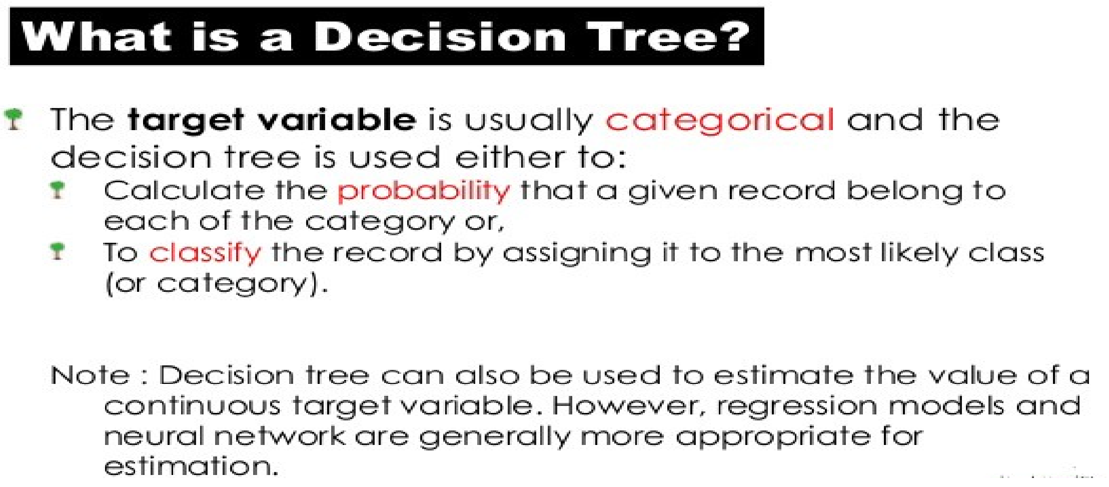
This is where decision tree helps, it will segregate the students based on all values of three variable and identify the variable, which creates the best homogeneous sets of students (which are heterogeneous to each other). In the snapshot below, you can see that variable Gender is able to identify best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

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. Example: - Let's say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that income of customer is a significant variable but insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree to predict customer income based on occupation, product and various other variables. In this case, we are predicting values for continuous variable



What is a Decision Tree?

- The **target variable** is usually categorical and the decision tree is used either to:
  - Calculate the probability that a given record belong to each of the category or,
  - To classify the record by assigning it to the most likely class (or category).

Note : Decision tree can also be used to estimate the value of a continuous target variable. However, regression models and neural network are generally more appropriate for estimation.

Important Terminology related to Decision Trees

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.

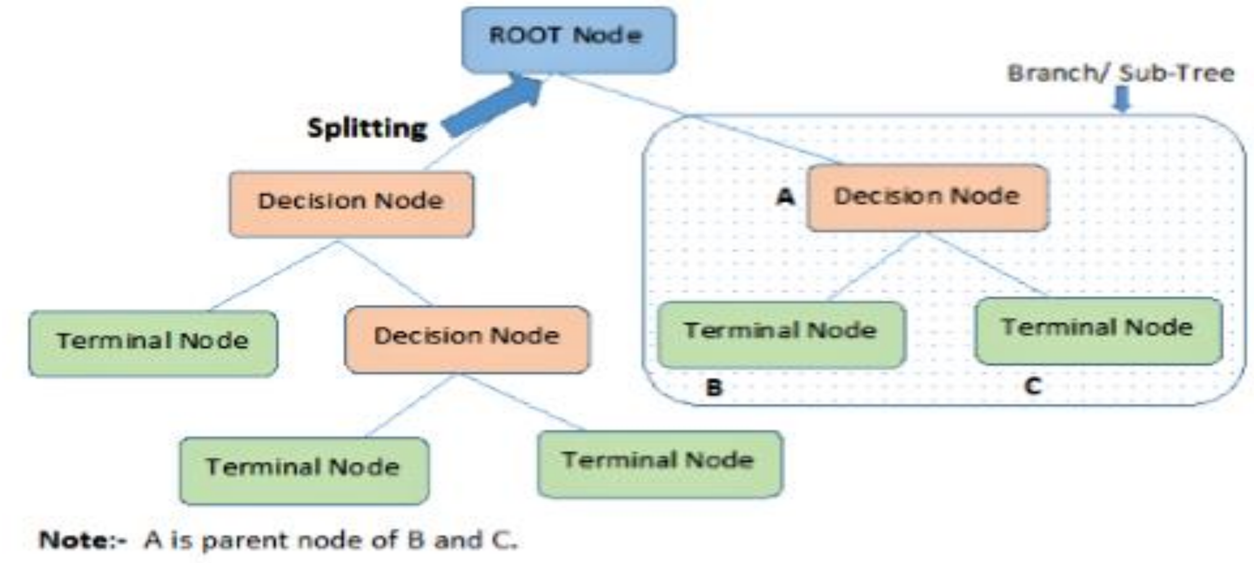
3. **Decision Node**: When a sub-node splits into further sub-nodes, then it is called decision node.

4. **Leaf/ Terminal Node**: Nodes do not split is called Leaf or Terminal node.

5. **Pruning**: When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.

6. **Branch / Sub-Tree**: A sub section of entire tree is called branch or sub-tree.

7. **Parent and Child Node**: A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node



Advantages

1. Easy to Understand: Decision tree output is very easy to understand even for people from non-analytical background. It does not require any statistical knowledge to read and interpret them. Its graphical representation is very intuitive and users can easily relate their hypothesis.

2. Useful in Data exploration: Decision tree is one of the fastest Algorithm to identify most significant variables and relation between two or more variables. With the help of decision trees, we can create new variables / features that has better power to predict target variable. It can also be used in data exploration stage. For example, we are working on a problem where we have information available in hundreds of variables, there decision tree will help to identify most significant variable.

3. Less data cleaning required: It requires less data cleaning compared to some other modeling techniques. It is not influenced by outliers and missing values to a fair degree.

4. Data type is not a constraint: It can handle both numerical and categorical variables.

5. Non-Parametric Method: Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

Disadvantages

1. Over fitting: Over fitting is one of the most practical difficulty for decision tree models. This problem gets solved by setting constraints on model parameters and pruning

2. Not fit for continuous variables: While working with continuous numerical variables, decision tree Loses information when it categorizes variables in different categories

The decision tree is one of the more popular classification algorithms being used in Data Mining and Machine Learning. applications include:

· Evaluation of brand expansion opportunities for a business using historical sales data

· Determination of likely buyers of a product using demographic data to enable targeting of limited advertisement budget

· Prediction of likelihood of default for applicant borrowers using predictive models generated from historical data

· Help with prioritization of emergency room patient treatment using a predictive model based on factors such as age, blood pressure, gender, location and severity of pain, and other measurements

· Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal.

Because of their simplicity, tree diagrams have been used in a broad range of industries and disciplines including civil planning, energy, financial, engineering, healthcare, pharmaceutical, education, law, and business.

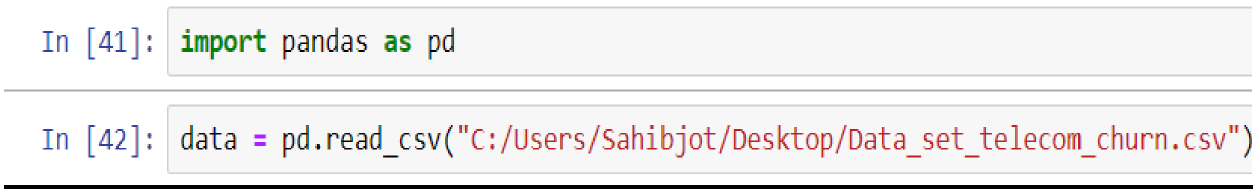
Assumptions while creating Decision Tree

Some of the assumptions we make while using Decision tree:

- At the beginning, the whole training set is considered as the **root.**
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are **distributed recursively** on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

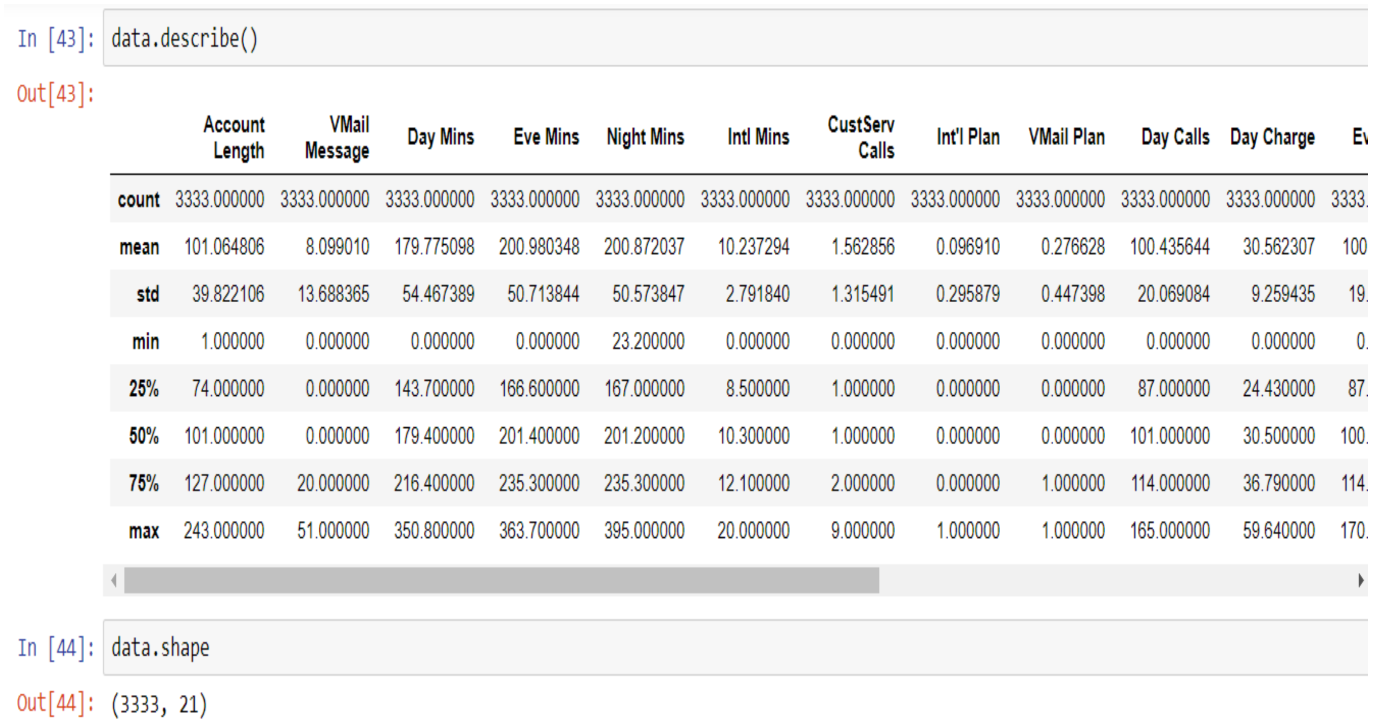
Importing Required Libraries and data

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")
```

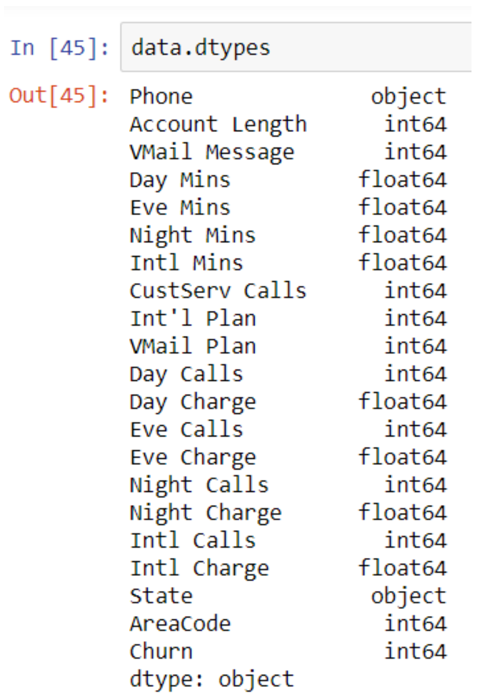


```
In [43]: data.describe()
```

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
```

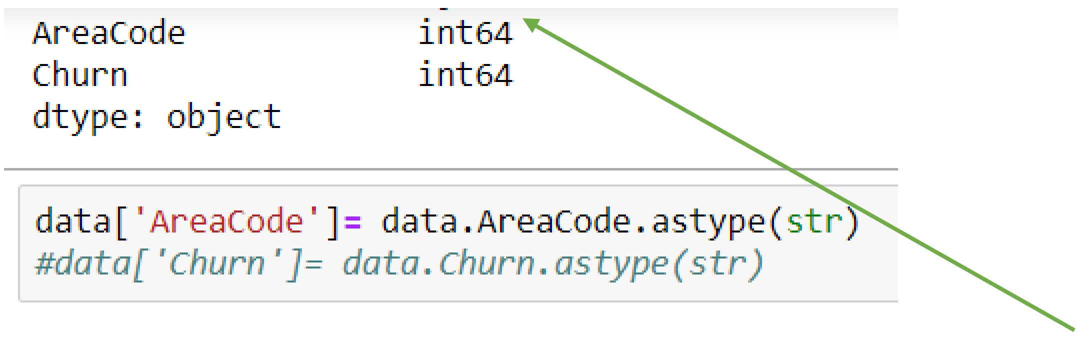
to see some of the core statistics about a particular column, you can use the 'describe' function.

- 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.



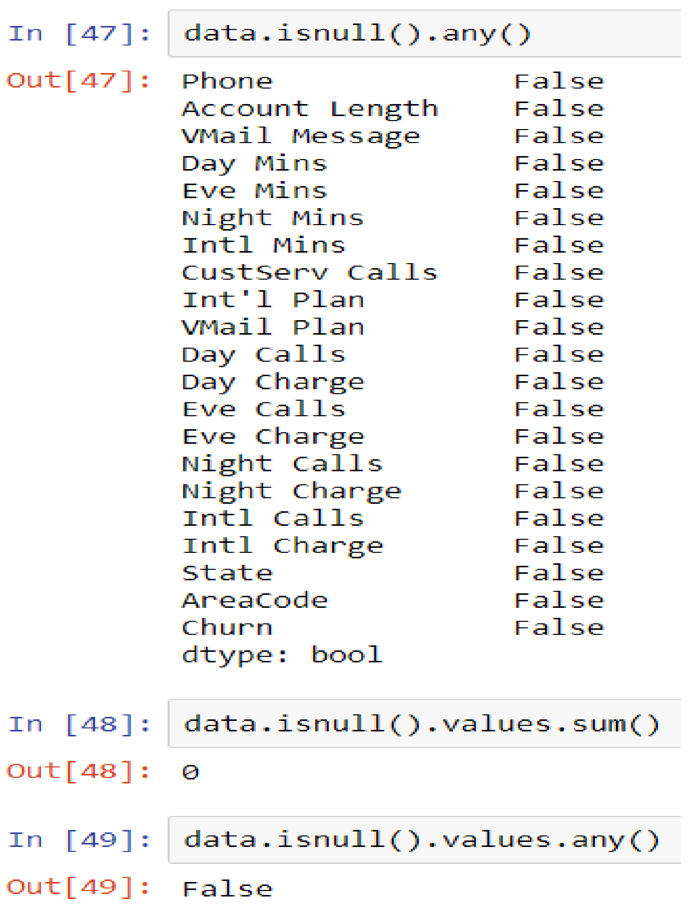
```
AreaCode            int64
Churn               int64
dtype: object
```

```
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](#)

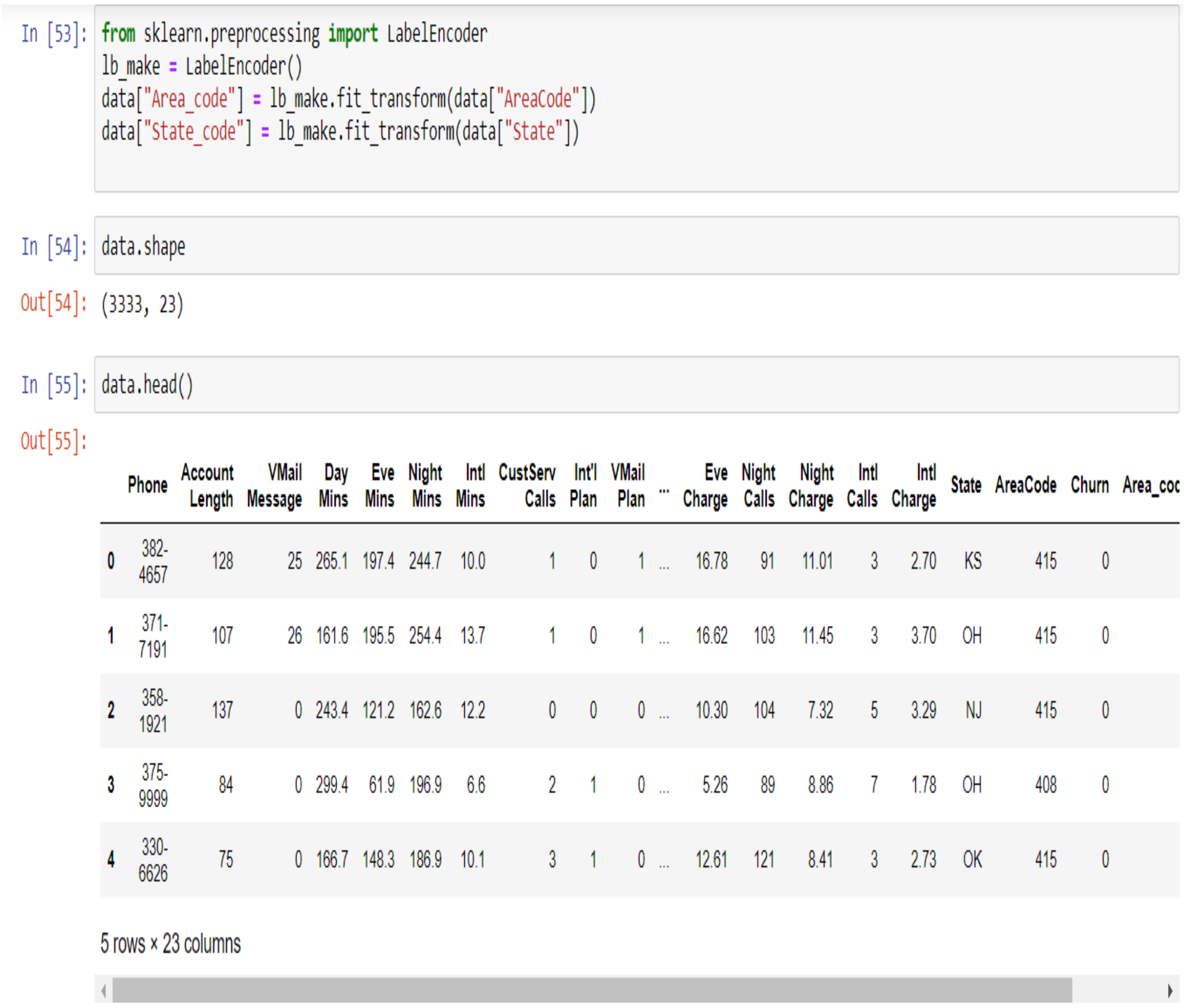


```
In [47]: data.isnull().any()

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.

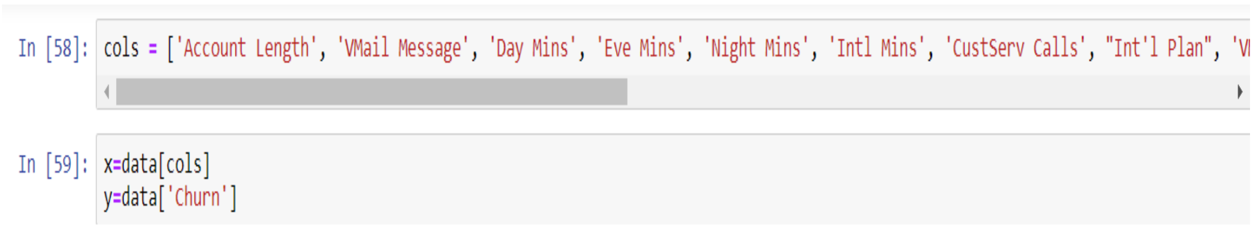


```
In [53]: from sklearn.preprocessing import LabelEncoder
         lb_make = LabelEncoder()
         data["Area_code"] = lb_make.fit_transform(data["AreaCode"])
         data["State_code"] = lb_make.fit_transform(data["State"])

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	

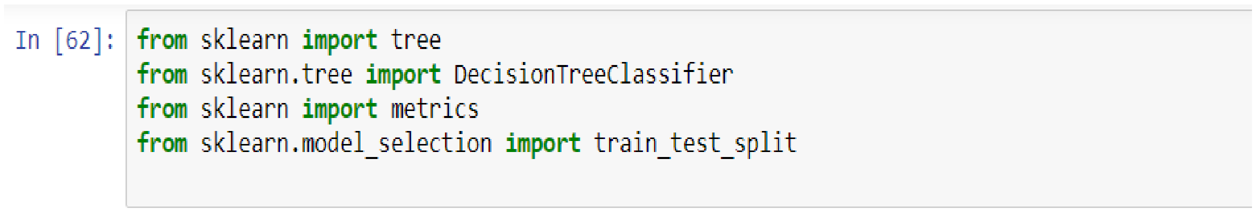
5 rows × 23 columns

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

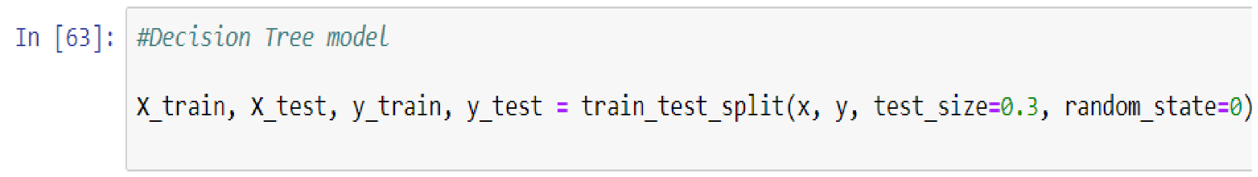


```
In [58]: cols = ['Account Length', 'VMail Message', 'Day Mins', 'Eve Mins', 'Night Mins', 'Intl Mins', 'CustServ Calls', "Int'l Plan", 'V

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.

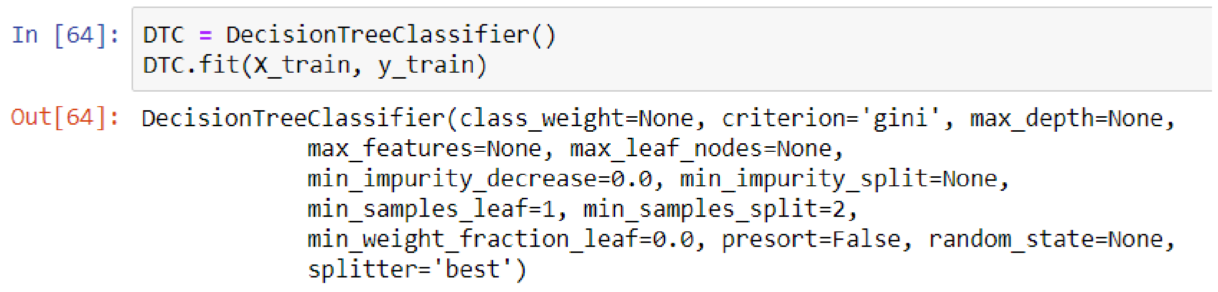
You can't possibly manually split the dataset into two. And you also have to make sure you split the data in a random manner. To help us with this task, the SciKit library provides a tool, called the Model Selection library. There's a class in the library which is, aptly, named 'train_test_split.' Using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



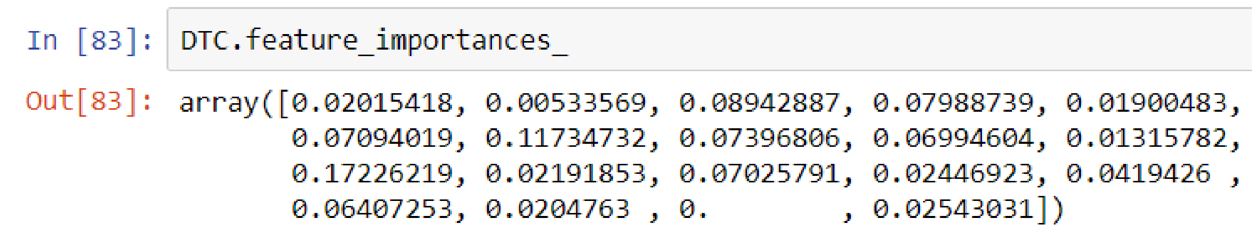
```
In [64]: DTC = DecisionTreeClassifier()
         DTC.fit(X_train, y_train)

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')
```

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.

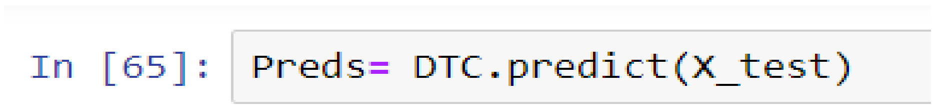
Importance is calculated for a single decision tree by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for. The performance measure may be the purity (Gini index) used to select the split points or another more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

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



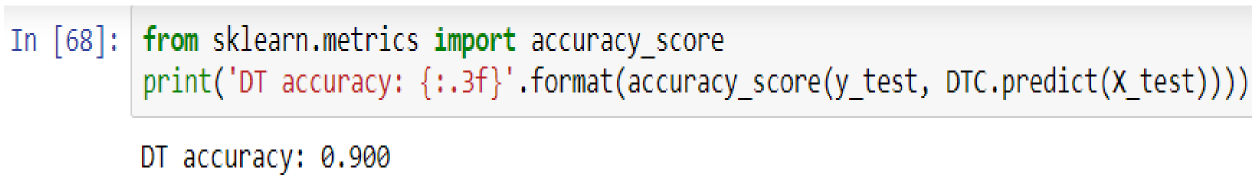
```
In [83]: DTC.feature_importances_

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:



```
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.

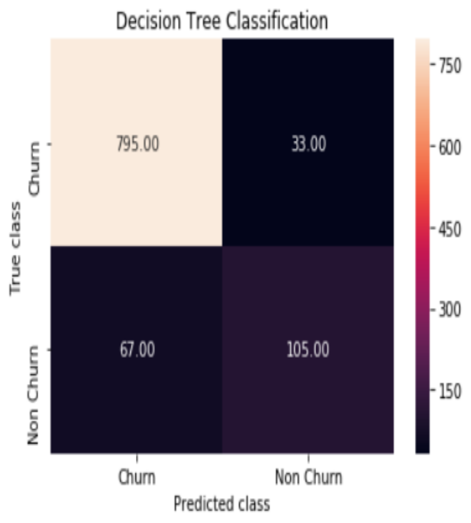
At this point we have trained our algorithm and made some predictions. Now we'll see how accurate our algorithm is. For classification tasks some commonly used metrics are confusion matrix, precision, recall, and F1 score. Lucky for us Scikit=-Learn's metrics library contains the classification_report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


Decision Tree Classification
Churn
Non Churn
True class
Churn
Non Churn
Predicted class