

# PYTHON TUTORIAL FOR BEGINNERS: LEARN IN 3 DAYS

This tutorial helps you to get started with Python. It's a step by step practical guide to learn Python by examples. Python is an open source language and it is widely used as a high-level programming language for general-purpose programming. It has gained high popularity in data science world. In the PyPL Popularity of Programming language index, Python scored second rank with a 14 percent share. In advanced analytics and predictive analytics market, it is ranked among top 3 programming languages for advanced analytics.

## Table of Contents

- 1. Getting Started with Python**
  - Python 2.7 vs. 3.6
  - Python for Data Science
  - How to install Python?
  - Spyder Shortcut keys
  - Basic programs in Python
  - Comparison, Logical and Assignment Operators
- 2. Data Structures and Conditional Statements**
  - Python Data Structures
  - Python Conditional Statements
- 3. Python Libraries**
  - List of popular packages (comparison with R)
  - Popular python commands
  - How to import a package
- 4. Data Manipulation using Pandas**
  - Pandas Data Structures - Series and DataFrame
  - Important Pandas Functions (vs. R functions)
  - Examples - Data analysis with Pandas
- 5. Data Science with Python**
  - Logistic Regression
  - Decision Tree
  - Random Forest
  - Grid Search - Hyper Parameter Tuning
  - Cross Validation
  - Preprocessing Steps

# Python 2.7 vs 3.6

Google yields thousands of articles on this topic. Some bloggers opposed and some in favor of 2.7. If you filter your search criteria and look for only recent articles (late 2016 onwards), you would see majority of bloggers are in favor of Python 3.6. **See the following reasons to support Python 3.6.**

1. The official end date for the Python 2.7 is year 2020. Afterward there would be no support from community. It does not make any sense to learn 2.7 if you learn it today.
2. Python 3.6 supports 95% of top 360 python packages and almost 100% of top packages for data science.

## What's new in Python 3.6

It is cleaner and faster. It is a language for the future. It fixed major issues with versions of Python 2 series. Python 3 was first released in year 2008. It has been 9 years releasing robust versions of Python 3 series.

## Key Takeaway

*You should go for Python 3.6. In terms of learning Python, there are no major differences in Python 2.7 and 3.6. It is not too difficult to move from Python 3 to Python 2 with a few adjustments. Your focus should go on learning Python as a language.*

## Python for Data Science

Python is widely used and very popular for a variety of software engineering tasks such as website development, cloud-architecture, back-end etc. It is equally popular in data science world. In advanced analytics world, there has been several debates on R vs. Python. There are some areas such as number of libraries for statistical analysis, where R wins over Python but Python is catching up very fast. With popularity of big data and data science, Python has become first programming language of data scientists.

There are several reasons to learn Python. Some of them are as follows –

1. Python runs well in automating various steps of a predictive model.
2. Python has awesome robust libraries for machine learning, natural language processing, deep learning, big data and artificial Intelligence.
3. Python wins over R when it comes to deploying machine learning models in production.
4. It can be easily integrated with big data frameworks such as Spark and Hadoop.
5. Python has a great online community support.

*Do you know these sites are developed in Python?*

1. YouTube
2. Instagram
3. Reddit

4. Dropbox
5. Disqus

## How to Install Python

There are two ways to download and install Python

1. [Download Anaconda](#). It comes with Python software along with preinstalled popular libraries.
2. Download [Python](#) from its official website. You have to manually install libraries.
- 3.

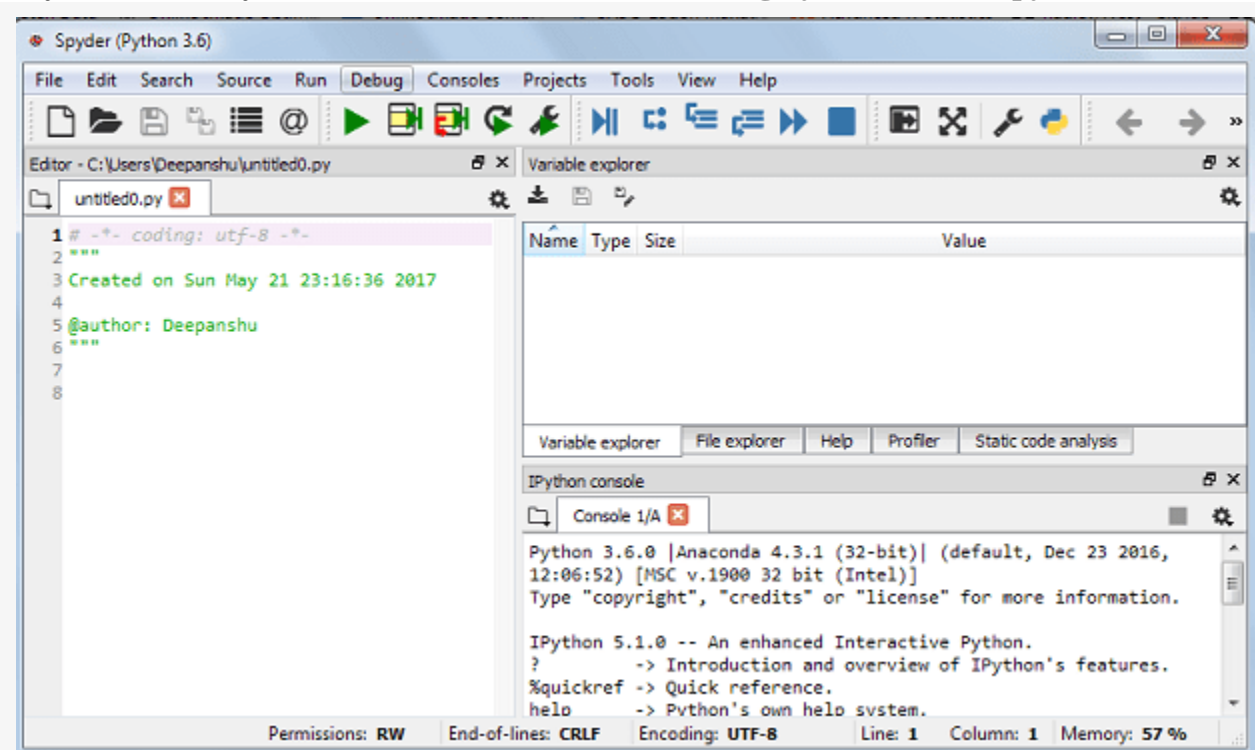
**Recommended :** Go for first option and download anaconda. It saves a lot of time in learning and coding Python

## Coding Environments

Anaconda comes with two popular IDE :

1. Jupyter (Ipython) Notebook
2. Spyder

**Spyder.** It is like RStudio for Python. It gives an environment wherein writing python code is user-friendly. If you are a SAS User, you can think of it as SAS Enterprise Guide / SAS Studio. It comes with a syntax editor where you can write programs. It has a console to check each and every line of code. Under the 'Variable explorer', you can access your created data files and function. **I highly recommend Spyder!**



Spyder - Python Coding Environment

## Jupyter (Ipython) Notebook

Jupyter is equivalent to markdown in R. It is useful when you need to present your work to others or when you need to create step by step project report as it can combine code, output, words, and graphics.

# Spyder Shortcut Keys

The following is a list of some useful spyder shortcut keys which makes you more productive.

1. **Press F5** to run the entire script
2. **Press F9** to run selection or line
3. Press **Ctrl + 1** to comment / uncomment
4. Go to front of function and then **press Ctrl + I** to see documentation of the function
5. **Run %reset -f** to clean workspace
6. **Ctrl + Left click on object** to see source code
7. **Ctrl+Enter** executes the current cell.
8. **Shift+Enter** executes the current cell and advances the cursor to the next cell

## *List of arithmetic operators with examples*

Arithmetic Operators	Operation	Example
+	Addition	$10 + 2 = 12$
−	Subtraction	$10 - 2 = 8$
*	Multiplication	$10 * 2 = 20$
/	Division	$10 / 2 = 5.0$
%	Modulus (Remainder)	$10 \% 3 = 1$
**	Power	$10 ** 2 = 100$
//	Floor	$17 // 3 = 5$
$(x + (d-1)) // d$	Ceiling	$(17 + (3-1)) // 3 = 6$

# Basic Programs

## Example 1

```
#Basics
x = 10
y = 3
print("10 divided by 3 is", x/y)
print("remainder after 10 divided by 3 is", x%y)
```

**Result :**

10 divided by 3 is 3.33

remainder after 10 divided by 3 is 1

## Example 2

```
x = 100
x > 80 and x <= 95
x > 35 or x < 60
```

```
x > 80 and x <=95
```

```
Out[45]: False
```

```
x > 35 or x < 60
```

```
Out[46]: True
```

Comparison & Logical Operators	Description	Example
>	Greater than	5 > 3 returns True
<	Less than	5 < 3 returns False
>=	Greater than or equal to	5 >= 3 returns True
<=	Less than or equal to	5 <= 3 return False
==	Equal to	5 == 3 returns False
!=	Not equal to	5 != 3 returns True
and	Check both the conditions	x > 18 and x <=35
or	If atleast one condition hold True	x > 35 or x < 60
not	Opposite of Condition	not(x>7)

## Assignment Operators

It is used to assign a value to the declared variable. For e.g. **x += 25 means x = x +25.**

```
x = 100
```

```
y = 10
```

```
x += y
```

```
print(x)
```

```
print(x)
```

```
110
```

In this case, x+=y implies x=x+y which is x = 100 + 10.

*Similarly, you can use x-=y, x\*=y and x /=y*

# Python Data Structure

In every programming language, it is important to understand the data structures. Following are some data structures used in Python.

## 1. List

It is a sequence of multiple values. It allows us to store different types of data such as integer, float, string etc. See the examples of list below. First one is an integer list containing only integer. Second one is string list containing only string values. Third one is mixed list containing integer, string and float values.

1. `x = [1, 2, 3, 4, 5]`
2. `y = ['A', 'O', 'G', 'M']`
3. `z = ['A', 4, 5.1, 'M']`

## Get List Item

We can extract list item using Indexes. **Index starts from 0 and end with (number of elements-1).**

```
x = [1, 2, 3, 4, 5]
x[0]
x[1]
x[4]
x[-1]
x[-2]
```

```
x[0]

Out[68]: 1


x[1]

Out[69]: 2


x[4]

Out[70]: 5


x[-1]

Out[71]: 5


x[-2]

Out[72]: 4
```

**x[0]** picks first element from list. **Negative sign** tells Python to search list item from right to left. **x[-1]** selects the last element from list.

You can select multiple elements from a list using the following method

```
x[:3] returns [1, 2, 3]
```

## 2. Tuple

A tuple is similar to a list in the sense that it is a sequence of elements. The difference between list and tuple are as follows -

1. A tuple cannot be changed once created whereas list can be modified.
2. A tuple is created by placing comma-separated values inside parentheses ( ). Whereas, list is created inside square brackets [ ]

### Examples

```
K = (1,2,3)  
City = ('Delhi','Mumbai','Bangalore')
```

### Perform for loop on Tuple

```
for i in City:  
    print(i)
```

```
Delhi  
  
Mumbai  
  
Bangalore
```

## Functions

Like **print()**, you can create your own custom function. It is also called user-defined functions. It helps you in automating the repetitive task and calling reusable code in easier way.

### Rules to define a function

1. Function starts with **def** keyword followed by function name and ( )
2. Function body starts with a colon (:) and is indented
3. The keyword **return** ends a function and give value of previous expression.

```
def sum_fun(a, b):  
    result = a + b  
    return result
```

```
z = sum_fun(10, 15)
```

**Result :  $z = 25$**

Suppose you want python to assume **0 as default value** if no value is specified for parameter b.

```
def sum_fun(a, b=0):  
    result = a + b  
    return result  
z = sum_fun(10)
```

In the above function, b is set to be 0 if no value is provided for parameter b. It does not mean no other value than 0 can be set here. It can also be used as **z = sum\_fun(10, 15)**

## Conditional Statements (if else)

Conditional statements are commonly used in coding. It is IF ELSE statements. It can be read like : " if a condition holds true, then execute something. Else execute something else"

**Note :** The if and else statements ends with a colon :

### Example

```
k = 27  
if k%5 == 0:  
    print('Multiple of 5')  
else:  
    print('Not a Multiple of 5')
```

**Result :** Not a Multiple of 5

## Popular python packages for Data Analysis & Visualization

Some of the leading packages in Python along with equivalent libraries in R are as follows-

1. **pandas**. For data manipulation and data wrangling. A collections of functions to understand and explore data. It is counterpart of **dplyr** and **reshape2** packages in R.
2. **NumPy**. For numerical computing. It's a package for efficient array computations. It allows us to do some operations on an entire column or table in one line. It is roughly approximate to **Rcpp** package in R which eliminates the limitation of slow speed in R.
3. **Scipy**. For mathematical and scientific functions such as integration, interpolation, signal processing, linear algebra, statistics, etc. It is built on Numpy.
4. **Scikit-learn**. A collection of machine learning algorithms. It is built on Numpy and Scipy. It can perform all the techniques that can be done in R using **glm**, **knn**, **randomForest**, **rpart**, **e1071** packages.
5. **Matplotlib**. For data visualization. It's a leading package for graphics in Python. It is equivalent to **ggplot2** package in R.
6. **Statsmodels**. For statistical and predictive modeling. It includes various functions to explore data and generate descriptive and predictive analytics. It allows users to run descriptive statistics, methods to impute missing values, statistical tests and take table output to HTML format.
7. **pandasql**. It allows SQL users to write SQL queries in Python. It is very helpful for people who loves writing SQL queries to manipulate data. It is equivalent to **sqldf** package in R.



*Maximum of the above packages are already preinstalled in Spyder.*

## Comparison of Python and R Packages by Data Mining Task

Task	Python Package	R Package
IDE	Rodeo / Spyder	Rstudio
Data Manipulation	pandas	dplyr and reshape2
Machine Learning	Scikit-learn	glm, knn, randomForest, rpart, e1071
Data Visualization	ggplot + seaborn + bokeh	ggplot2
Character Functions	Built-In Functions	stringr
Reproducibility	Jupyter	Knitr
SQL Queries	pandasql	sqldf
Working with Dates	datetime	lubridate
Web Scraping	beautifulsoup	rvest

## Popular Python Commands

The commands below would help you to install and update new and existing packages. Let's say, you want to install / uninstall **pandas** package.

### Install Package

```
!pip install pandas
```

### Uninstall Package

```
!pip uninstall pandas
```

### Show Information about Installed Package

```
!pip show pandas
```

### List of Installed Packages

```
!pip list
```

### Upgrade a package

```
!pip install --upgrade pandas
```

## How to import a package

There are multiple ways to import a package in Python. It is important to understand the difference between these styles.

### 1. import pandas as pd

It imports the package **pandas** under the alias **pd**. A function **DataFrame** in package pandas is then submitted with **pd.DataFrame**.

### 2. import pandas

It imports the package without using alias but here the function **DataFrame** is submitted with full package name **pandas.DataFrame**

### 3. from pandas import \*

It imports the whole package and the function **DataFrame** is executed simply by typing **DataFrame**. It sometimes creates confusion when same function name exists in more than one package.

## Pandas Data Structures : Series and DataFrame

In pandas package, there are two data structures - series and dataframe. These structures are explained below in detail -

1. **Series** is a one-dimensional array. You can access individual elements of a series using position. It's similar to vector in R.

*In the example below, we are generating 5 random values.*

```
import pandas as pd
s1 = pd.Series(np.random.randn(5))
s1
```

```
0    -2.412015
1    -0.451752
2     1.174207
3     0.766348
4    -0.361815

dtype: float64
```

### Extract first and second value

You can get a particular element of a series using index value. See the examples below -

```
s1[0]
```

```
-2.412015
```

```
s1[1]
```

```
-0.451752
```

```
s1[:3]
```

```
0    -2.412015
```

1	-0.451752
2	1.174207

## 2. DataFrame

It is equivalent to data.frame in R. It is a 2-dimensional data structure that can store data of different data types such as characters, integers, floating point values, factors. Those who are well-conversant with MS Excel, they can think of data frame as Excel Spreadsheet.

## Comparison of Data Type in Python and Pandas

The following table shows how Python and pandas package stores data.

Data Type	Pandas	Standard Python
For character variable	object	string
For categorical variable	category	-
For Numeric variable without decimals	int64	int
Numeric characters with decimals	float64	float
For date time variables	datetime64	-

## Important Pandas Functions

The table below shows comparison of pandas functions with R functions for various data wrangling and manipulation tasks. It would help you to memorise pandas functions. It's a very handy information for programmers who are new to Python. It includes solutions for most of the frequently used data exploration tasks.

Functions	R	Python (pandas package)
Installing a package	install.packages('name')	!pip install name
Loading a package	library(name)	import name as other_name
Checking working directory	getwd()	import os os.getcwd()
Setting working directory	setwd()	os.chdir()
List files in a directory	dir()	os.listdir()
Remove an object	rm('name')	del object
Select Variables	select(df, x1, x2)	df[['x1', 'x2']]
Drop Variables	select(df, -(x1:x2))	df.drop(['x1', 'x2'], axis = 1)
Filter Data	filter(df, x1 >= 100)	df.query('x1 >= 100')
Structure of a DataFrame	str(df)	df.info()

Summarize dataframe	summary(df)	df.describe()
Get row names of dataframe "df"	rownames(df)	df.index
Get column names	colnames(df)	df.columns
View Top N rows	head(df,N)	df.head(N)
View Bottom N rows	tail(df,N)	df.tail(N)
Get dimension of data frame	dim(df)	df.shape
Get number of rows	nrow(df)	df.shape[0]
Get number of columns	ncol(df)	df.shape[1]
Length of data frame	length(df)	len(df)
Get random 3 rows from dataframe	sample_n(df, 3)	df.sample(n=3)
Get random 10% rows	sample_frac(df, 0.1)	df.sample(frac=0.1)
Check Missing Values	is.na(df\$x)	pd.isnull(df.x)
Sorting	arrange(df, x1, x2)	df.sort_values(['x1', 'x2'])
Rename Variables	rename(df, newvar = x1)	df.rename(columns={'x1': 'newvar'})

# Data Manipulation with pandas - Examples

## 1. Import Required Packages

You can import required packages using **import** statement. In the syntax below, we are asking Python to import numpy and pandas package. The 'as' is used to alias package name.

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

## 2. Build DataFrame

We can build dataframe using **DataFrame()** function of pandas package.

```
mydata = {'productcode': ['AA', 'AA', 'AA', 'BB', 'BB', 'BB'],
          'sales': [1010, 1025.2, 1404.2, 1251.7, 1160, 1604.8],
          'cost' : [1020, 1625.2, 1204, 1003.7, 1020, 1124]}
df = pd.DataFrame(mydata)
```

In this dataframe, we have three variables - productcode, sales, cost.

<i>productcode</i>	<i>sales</i>	<i>cost</i>
AA	1010	1020
AA	1025.2	1625.2
AA	1404.2	1204
BB	1251.7	1003.7
BB	1160	1020
BB	1604.8	1124

Sample DataFrame

## To import data from CSV file

You can use `read_csv()` function from pandas package to get data into python from CSV file.

```
mydata= pd.read_csv("C:\\Users\\Deepanshu\\Documents\\file1.csv")
```

Make sure you use **double backslash** when specifying path of CSV file. Alternatively, you can use forward slash to mention file path inside `read_csv()` function.

## Detailed Tutorial : [Import Data in Python](#)

### 3. To see number of rows and columns

You can run the command below to find out number of rows and columns.

```
df.shape
```

**Result :** (6, 3). It means 6 rows and 3 columns.

### 4. To view first 3 rows

The `df.head(N)` function can be used to check out first some N rows.

```
df.head(3)
```

	cost	productcode	sales
0	1020.0	AA	1010.0
1	1625.2	AA	1025.2
2	1204.0	AA	1404.2

### 5. Select or Drop Variables

To keep a single variable, you can write in any of the following three methods -

```
df.productcode
df["productcode"]
df.loc[:, "productcode"]
```

To select variable by column position, you can use **df.iloc** function. In the example below, we are selecting second column. Column Index starts from 0. Hence, 1 refers to second column.

```
df.iloc[:, 1]
```

We can keep multiple variables by specifying desired variables inside [ ]. Also, we can make use of df.loc() function.

```
df[["productcode", "cost"]]
df.loc[:, ["productcode", "cost"]]
```

## Drop Variable

We can remove variables by using df.drop() function. See the example below -

```
df2 = df.drop(['sales'], axis = 1)
```

## 6. To summarize data frame

To summarize or explore data, you can submit the command below.

```
df.describe()
```

	cost	sales
count	6.000000	6.00000
mean	1166.150000	1242.65000
std	237.926793	230.46669
min	1003.700000	1010.00000
25%	1020.000000	1058.90000
50%	1072.000000	1205.85000
75%	1184.000000	1366.07500
max	1625.200000	1604.80000

To summarise all the **character variables**, you can use the following script.

```
df.describe(include=['O'])
```

Similarly, you can use **df.describe(include=['float64'])** to view summary of all the numeric variables with decimals.

To select only a particular variable, you can write the following code -

```
df.productcode.describe()  
OR  
df["productcode"].describe()
```

```
count      6  
  
unique      2  
  
top         BB  
  
freq        3  
  
Name: productcode, dtype: object
```

## 7. To calculate summary statistics

We can manually find out summary statistics such as count, mean, median by using commands below

```
df.sales.mean()  
df.sales.median()  
df.sales.count()  
df.sales.min()  
df.sales.max()
```

## 8. Filter Data

Suppose you are asked to apply condition - productcode is equal to "AA" and sales greater than or equal to 1250.

```
df1 = df[(df.productcode == "AA") & (df.sales >= 1250)]
```

It can also be written like :

```
df1 = df.query('(productcode == "AA") & (sales >= 1250)')
```

In the second query, we **do not need** to specify DataFrame along with variable name.

## 9. Sort Data

In the code below, we are arrange data in ascending order by sales.

```
df.sort_values(['sales'])
```

## 10. Group By : Summary by Grouping Variable

Like SQL GROUP BY, you want to summarize continuous variable by classification variable. In this case, we are calculating average sale and cost by product code.

```
df.groupby(df.productcode).mean()
```

	cost	sales
productcode		
AA	1283.066667	1146.466667
BB	1049.233333	1338.833333

Instead of summarising for multiple variable, you can run it for a single variable i.e. sales. Submit the following script.

```
df["sales"].groupby(df.productcode).mean()
```

## 11. Define Categorical Variable

Let's create a classification variable - id which contains only 3 unique values - 1/2/3.

```
df0 = pd.DataFrame({'id': [1, 1, 2, 3, 1, 2, 2]})
```

*Let's define as a categorical variable.*

We can use astype() function to make id as a categorical variable.

```
df0.id = df0["id"].astype('category')
```

Summarize this classification variable to check descriptive statistics.

```
df0.describe()
```

	id
count	7
unique	3
top	2
freq	3



## Frequency Distribution

You can calculate frequency distribution of a categorical variable. It is one of the method to explore a categorical variable.

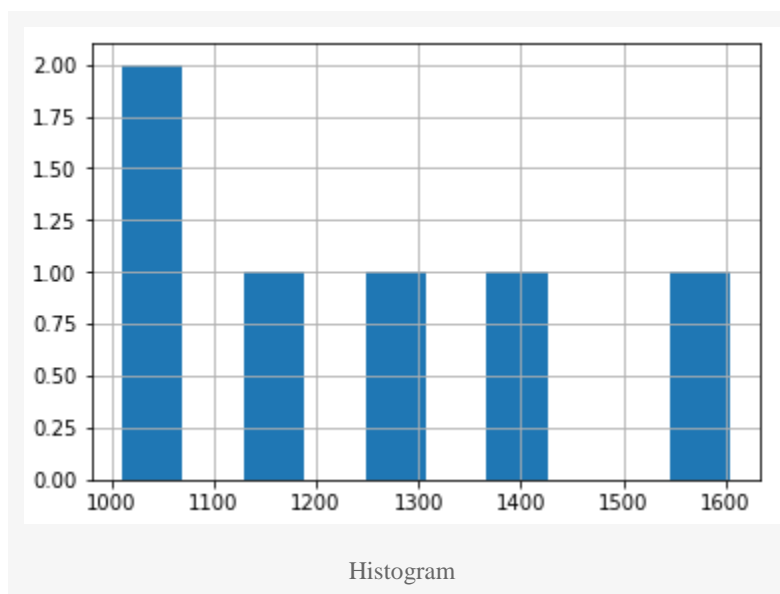
```
df['productcode'].value_counts()
```

BB	3
AA	3

## 12. Generate Histogram

Histogram is one of the method to check distribution of a continuous variable. In the figure shown below, there are two values for variable 'sales' in range 1000-1100. In the remaining intervals, there is only a single value. In this case, there are only 5 values. If you have a large dataset, you can plot histogram to identify outliers in a continuous variable.

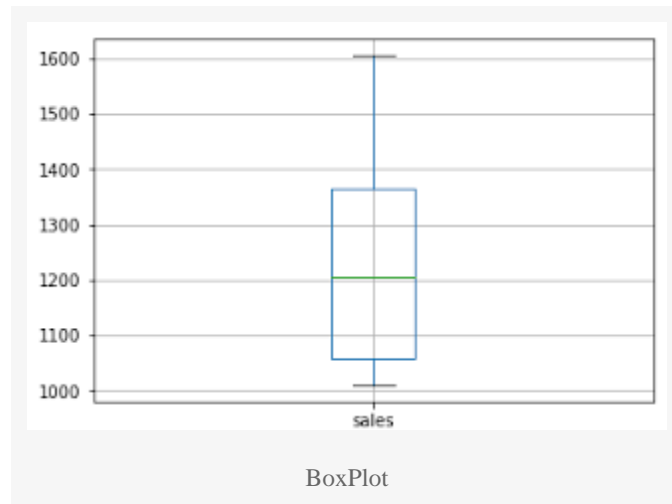
```
df['sales'].hist()
```



## 13. BoxPlot

Boxplot is a method to visualize continuous or numeric variable. It shows minimum, Q1, Q2, Q3, IQR, maximum value in a single graph.

```
df.boxplot(column='sales')
```



## Data Science using Python - Examples

In this section, we cover how to perform data mining and machine learning algorithms with Python. sklearn is the most frequently used library for running data mining and machine learning algorithms. We will also cover statsmodels library for regression techniques. statsmodels library generates formattable output which can be used further in project report and presentation.

### 1. Install the required libraries

Import the following libraries before reading or exploring data

```
#Import required libraries
import pandas as pd
import statsmodels.api as sm
import numpy as np
```

### 2. Download and import data into Python

With the use of python library, we can easily get data from web into python.

```
# Read data from web
df = pd.read_csv("https://stats.idre.ucla.edu/stat/data/binary.csv")
```

Variables	Type	Description
gre	Continuous	Graduate Record Exam score
gpa	Continuous	Grade Point Average
rank	Categorical	Prestige of the undergraduate institution
admit	Binary	Admission in graduate school

The binary variable admit is a target variable.

### 3. Explore Data

Let's explore data. We'll answer the following questions -

1. How many rows and columns in the data file?
2. What are the distribution of variables?
3. Check if any outlier(s)
4. If outlier(s), treat them
5. Check if any missing value(s)
6. Impute Missing values (if any)

```
# See no. of rows and columns  
df.shape
```

**Result :** 400 rows and 4 columns

In the code below, we rename the variable rank to '**position**' as rank is already a function in python.

```
# rename rank column  
df = df.rename(columns={'rank': 'position'})
```

Summarize and plot all the columns.

```
# Summarize  
df.describe()  
# plot all of the columns  
df.hist()
```

## Categorical variable Analysis

It is important to check the frequency distribution of categorical variable. It helps to answer the question whether data is skewed.

```
# Summarize  
df.position.value_counts(ascending=True)
```

1	61
4	67
3	121
2	151

## Generating Crosstab

By looking at cross tabulation report, we can check whether we have enough number of events against each unique values of categorical variable.

```
pd.crosstab(df['admit'], df['position'])
```

position	1	2	3	4
admit				
0	28	97	93	55
1	33	54	28	12

## Number of Missing Values

We can write a simple loop to figure out the number of blank values in all variables in a dataset.

```
for i in list(df.columns):
    k = sum(pd.isnull(df[i]))
    print(i, k)
```

In this case, there are no missing values in the dataset.

## 4. Logistic Regression Model

Logistic Regression is a special type of regression where target variable is categorical in nature and independent variables be discrete or continuous. In this post, we will demonstrate only **binary logistic regression** which takes only binary values in target variable. Unlike linear regression, logistic regression model returns probability of target variable. It assumes binomial distribution of dependent variable. In other words, it belongs to binomial family.

In python, we can write R-style model formula  $y \sim x_1 + x_2 + x_3$  using **patsy** and **statsmodels** libraries. In the formula, we need to define variable 'position' as a categorical variable by mentioning it inside capital C(). You can also define reference category using **reference=** option.

```
#Reference Category
from patsy import dmatrices, Treatment
y, X = dmatrices('admit ~ gre + gpa + C(position, Treatment(reference=4))', df, return_type = 'dataframe')
```

It returns two datasets - X and y. The dataset 'y' contains variable admit which is a target variable. The other dataset 'X' contains Intercept (constant value), dummy variables for Treatment, gre and gpa. Since 4 is set as a reference category, it will be 0 against all the three dummy variables. See sample below -

P	P_1	P_2	P_3
3	0	0	1
3	0	0	1
1	1	0	0
4	0	0	0

4 0 0 0

2 0 1 0

## Split Data into two parts

80% of data goes to training dataset which is used for building model and 20% goes to test dataset which would be used for validating the model.

```
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)
```

## Build Logistic Regression Model

By default, the **regression without formula style** does not include intercept. To include it, we already have added **intercept** in **X\_train** which would be used as a predictor.

```
#Fit Logit model
logit = sm.Logit(y_train, X_train)
result = logit.fit()

#Summary of Logistic regression model
result.summary()
result.params
```

### Logit Regression Results

```
=====
Dep. Variable:          admit    No. Observations:          320
Model:                  Logit    Df Residuals:              315
Method:                  MLE     Df Model:                  4
Date:                   Sat, 20 May 2017    Pseudo R-squ.:        0.03399
Time:                   19:57:24    Log-Likelihood:        -193.49
converged:              True     LL-Null:              -200.30
                               LLR p-value:          0.008627
=====
```

```
coef    std err          z      P|z|    [95.0% Conf. Int.]
```

C(position)[T.1]	1.4933	0.440	3.392	0.001	0.630	2.356
C(position)[T.2]	0.6771	0.373	1.813	0.070	-0.055	1.409
C(position)[T.3]	0.1071	0.410	0.261	0.794	-0.696	0.910
gre	0.0005	0.001	0.442	0.659	-0.002	0.003
gpa	0.4613	0.214	-2.152	0.031	-0.881	-0.041

## Confusion Matrix and Odd Ratio

Odd ratio is exponential value of parameter estimates.

```
#Confusion Matrix
result.pred_table()
#Odd Ratio
np.exp(result.params)
```

## Prediction on Test Data

In this step, we take estimates of logit model which was built on training data and then later apply it into test data.

```
#prediction on test data
y_pred = result.predict(X_test)
```

## Calculate Area under Curve (ROC)

```
# AUC on test data
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
auc(false_positive_rate, true_positive_rate)
```

**Result :** AUC = 0.6763

## Calculate Accuracy Score

```
accuracy_score([ 1 if p > 0.5 else 0 for p in y_pred ], y_test)
```

# Decision Tree Model

Decision trees can have a target variable continuous or categorical. When it is continuous, it is called regression tree. And when it is categorical, it is called classification tree. It selects a variable at each step that best splits the set of values. There are several algorithms to find best split. Some of them are Gini, Entropy, C4.5, Chi-Square. There are several advantages of decision tree. It is simple to use and easy to understand. It requires a very few data preparation steps. It can handle mixed data - both categorical and continuous variables. In terms of speed, it is a very fast algorithm.

```
#Drop Intercept from predictors for tree algorithms

X_train = X_train.drop(['Intercept'], axis = 1)

X_test = X_test.drop(['Intercept'], axis = 1)


#Decision Tree

from sklearn.tree import DecisionTreeClassifier

model_tree = DecisionTreeClassifier(max_depth=7)


#Fit the model:

model_tree.fit(X_train,y_train)


#Make predictions on test set

predictions_tree = model_tree.predict_proba(X_test)


#AUC

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
predictions_tree[:,1])

auc(false_positive_rate, true_positive_rate)
```

**Result :** AUC = 0.664

## Important Note

*Feature engineering plays an important role in building predictive models. In the above case, we have not performed variable selection. We can also select best parameters by using grid search fine tuning technique.*

## Random Forest Model

Decision Tree has limitation of overfitting which implies it does not generalize pattern. It is very sensitive to a small change in training data. To overcome this problem, random forest comes into picture. It grows a large number of trees on randomised data. It selects random number of variables to grow each tree. It is more robust algorithm than decision tree. It is one of the most popular machine learning algorithm. It is commonly used in data science competitions. It is always ranked in top 5 algorithms. It has become a part of every data science toolkit.

```
#Random Forest

from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier(n_estimators=100, max_depth=7)

#Fit the model:

target = y_train['admit']

model_rf.fit(X_train,target)

#Make predictions on test set

predictions_rf = model_rf.predict_proba(X_test)

#AUC

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
predictions_rf[:,1])

auc(false_positive_rate, true_positive_rate)

#Variable Importance

importances = pd.Series(model_rf.feature_importances_,
index=X_train.columns).sort_values(ascending=False)
```



```
print(importances)

importances.plot.bar()
```

**Result :** AUC = 0.6974

## Grid Search - Hyper Parameters Tuning

The sklearn library makes hyper-parameters tuning very easy. It is a strategy to select the best parameters for an algorithm. In scikit-learn they are passed as arguments to the constructor of the estimator classes. For example, max\_features in randomforest. alpha for lasso.

```
from sklearn.model_selection import GridSearchCV

rf = RandomForestClassifier()

target = y_train['admit']

param_grid = {

    'n_estimators': [100, 200, 300],

    'max_features': ['sqrt', 3, 4]

}

CV_rfc = GridSearchCV(estimator=rf , param_grid=param_grid, cv= 5, scoring='roc_auc')

CV_rfc.fit(X_train,target)

#Parameters with Scores

CV_rfc.grid_scores_

#Best Parameters

CV_rfc.best_params_
```

```

CV_rfc.best_estimator_

#Make predictions on test set

predictions_rf = CV_rfc.predict_proba(X_test)

#AUC

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
predictions_rf[:,1])

auc(false_positive_rate, true_positive_rate)

```

## Cross Validation

```

# Cross Validation
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict, cross_val_score
target = y['admit']
prediction_logit = cross_val_predict(LogisticRegression(), X, target, cv=10, method='predict_proba')
#AUC
cross_val_score(LogisticRegression(fit_intercept = False), X, target, cv=10, scoring='roc_auc')

```

## Data Mining : PreProcessing Steps

1. The machine learning package **sklearn** requires all categorical variables in numeric form. Hence, we need to convert all character/categorical variables to be numeric. This can be accomplished using the following script. In sklearn, there is already a function for this step.

```

from sklearn.preprocessing import LabelEncoder

def ConverttoNumeric(df):

    cols = list(df.select_dtypes(include=['category', 'object']))

    le = LabelEncoder()

    for i in cols:

        try:

```

```

df[i] = le.fit_transform(df[i])

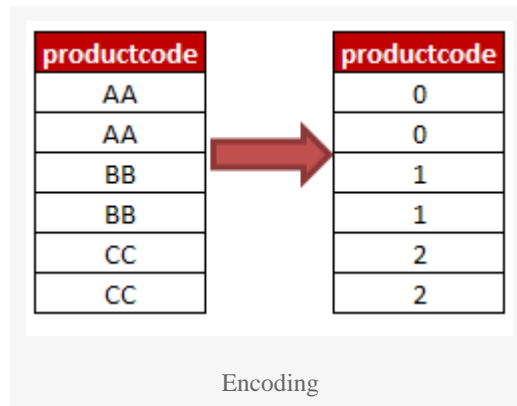
except:

    print('Error in variable :'+i)

return df

```

```
ConverttoNumeric(mydf)
```



## 2. Impute Missing Values

Imputing missing values is an important step of predictive modeling. In many algorithms, if missing values are not filled, it removes complete row. If data contains a lot of missing values, it can lead to huge data loss. There are multiple ways to impute missing values. Some of the common techniques - to replace missing value with mean/median/zero. It makes sense to replace missing value with 0 when 0 signifies meaningful. For example, whether customer holds a credit card product.

```

# fill missing values with 0
df['var1'] = df['var1'].fillna(0)
# fill missing values with mean
df['var1'] = df['var1'].fillna(df['var1'].mean())

```

## 3. Outlier Treatment

There are many ways to handle or treat outliers (or extreme values). Some of the methods are as follows -

1. Cap extreme values at 95th / 99th percentile depending on distribution
2. Apply log transformation of variables. See below the implementation of log transformation in Python.

```

import numpy as np
df['var1'] = np.log(df['var1'])

```

## Next Steps

Practice, practice and practice. Download free public data sets from Kaggle / UCLA websites and try to play around with data and generate insights from it with pandas package and build statistical models using sklearn package. I hope you would find this tutorial helpful. I tried to cover all the important topics which beginner must know about Python. Once completion of this tutorial, you can flaunt you know how to program it in Python and you can implement machine learning algorithms using sklearn package.