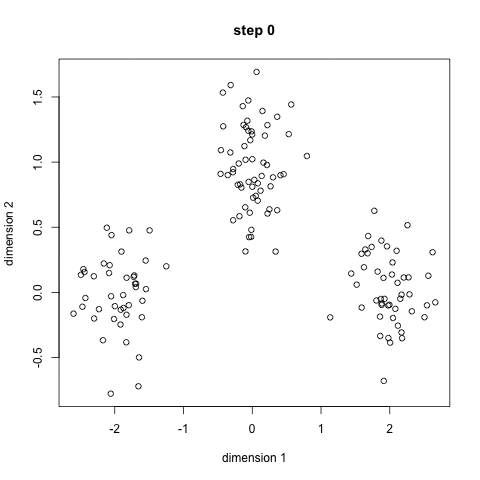
Clustering is a Machine Learning technique that involves the grouping of data points. Given a set of data points, we can use a clustering algorithm to classify each data point into a specific group. In theory, data points that are in the same group should have similar properties and/or features, while data points in different groups should have highly dissimilar properties and/or features. Clustering is a method of unsupervised learning and is a common technique for statistical data analysis used in many fields.

In Data Science, we can use clustering analysis to gain some val­uable insights from our data by seeing what groups the data points fall into when we apply a clustering algorithm. Today, we’re going to look at 5 popular clustering algorithms that data scientists need to know and their pros and cons!

**K-Means Clustering**

K-Means is probably the most well know clustering algorithm. It’s taught in a lot of introductory data science and machine learning classes. It’s easy to understand and implement in code! Check out the graphic below for an illustration.



K-Means Clustering

1. To begin, we first select a number of classes/groups to use and randomly initialize their respective center points. To figure out the number of classes to use, it’s good to take a quick look at the data and try to identify any distinct groupings. The center points are vectors of the same length as each data point vector and are the “X’s” in the graphic above.
2. Each data point is classified by computing the distance between that point and each group center, and then classifying the point to be in the group whose center is closest to it.
3. Based on these classified points, we recompute the group center by taking the mean of all the vectors in the group.
4. Repeat these steps for a set number of iterations or until the group centers don’t change much between iterations. You can also opt to randomly initialize the group centers a few times, and then select the run that looks like it provided the best results.

K-Means has the advantage that it’s pretty fast, as all we’re really doing is computing the distances between points and group centers; very few computations! It thus has a linear complexity *O*(*n*).

On the other hand, K-Means has a couple of disadvantages. Firstly, you have to select how many groups/classes there are. This isn’t always trivial and ideally with a clustering algorithm we’d want it to figure those out for us because the point of it is to gain some insight from the data. K-means also starts with a random choice of cluster centers and therefore it may yield different clustering results on different runs of the algorithm. Thus, the results may not be repeatable and lack consistency. Other cluster methods are more consistent.

K-Medians is another clustering algorithm related to K-Means, except instead of recomputing the group center points using the mean we use the median vector of the group. This method is less sensitive to outliers (because of using the Median) but is much slower for larger datasets as sorting is required on each iteration when computing the Median vector.

pandas as pd

Pandas is an open-source, BSD-licensed Python library providing high-performance,

easy-to-use data structures and data analysis tools for the Python programming language.

Python with Pandas is used in a wide range of fields including academic and

commercial domains including finance, economics, Statistics, analytics, etc.

What problem does pandas solve?

Python has long been great for data munging and preparation, but less so for data analysis and modeling.

pandas helps fill this gap, enabling you to carry out your entire data analysis workflow in Python

without having to switch to a more domain specific language like R.

Combined with the excellent IPython toolkit and other libraries,

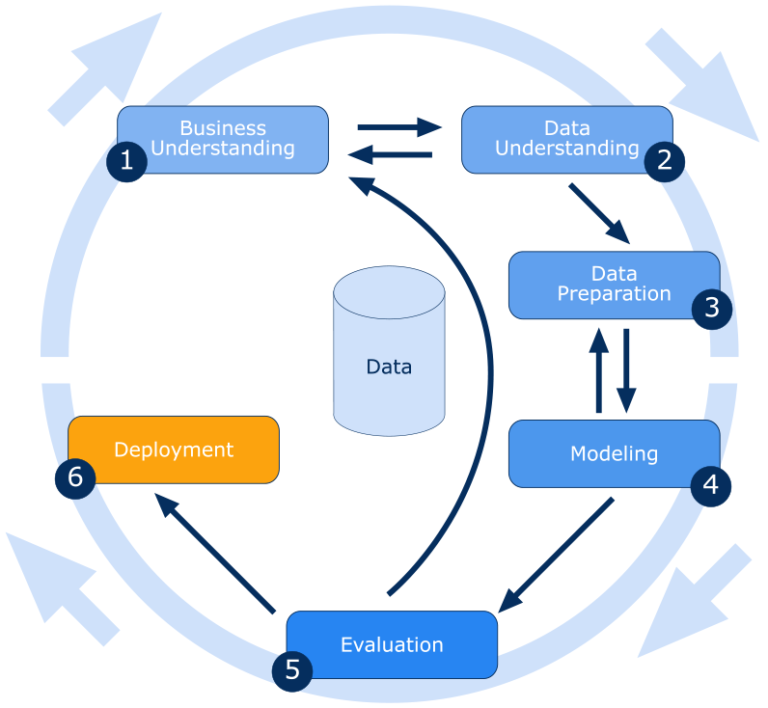
the environment for doing data analysis in Python excels in performance, productivity, and the ability to collaborate.

pandas does not implement significant modeling functionality outside of linear and panel regression;for this, look to statsmodels and scikit-learn.

More work is still needed to make Python a first class statistical modeling environment,

but we are well on our way toward that goal.

DATA Mining



Data mining process is the discovery through large data sets of patterns, relationships and insights that guide enterprises measuring and managing where they are and predicting where they will be in the future.

Large amount of data and databases can come from various data sources and may be stored in different data warehousess. And, data mining techniques such as machine learning, artificial intelligence (AI) and predictive modeling can be involved.

The data mining process requires commitment. But experts agree, across all industries, the data mining process is the same. And should follow a prescribed path.

Here are the 6 essential steps of the data mining process.

1. Business understanding

In the business understanding phase:

First, it is required to understand business objectives clearly and find out what are the business’s needs.

Next, assess the current situation by finding the resources, assumptions, constraints and other important factors which should be considered.

Then, from the business objectives and current situations, create data mining goals to achieve the business objectives within the current situation.

Finally, a good data mining plan has to be established to achieve both business and data mining goals. The plan should be as detailed as possible.

2. Data understanding

The data understanding phase starts with initial data collection, which is collected from available data sources, to help get familiar with the data. Some important activities must be performed including data load and data integration in order to make the data collection successfully.

Next, the “gross” or “surface” properties of acquired data need to be examined carefully and reported.

Then, the data needs to be explored by tackling the data mining questions, which can be addressed using querying, reporting, and visualization.

Finally, the data quality must be examined by answering some important questions such as “Is the acquired data complete?”, “Is there any missing values in the acquired data?”

3. Data preparation

The data preparation typically consumes about 90% of the time of the project. The outcome of the data preparation phase is the final data set. Once available data sources are identified, they need to be selected, cleaned, constructed and formatted into the desired form. The data exploration task at a greater depth may be carried during this phase to notice the patterns based on business understanding.

4. Modeling

First, modeling techniques have to be selected to be used for the prepared data set.

Next, the test scenario must be generated to validate the quality and validity of the model.

Then, one or more models are created on the prepared data set.

Finally, models need to be assessed carefully involving stakeholders to make sure that created models are met business initiatives.

5. Evaluation

In the evaluation phase, the model results must be evaluated in the context of business objectives in the first phase. In this phase, new business requirements may be raised due to the new patterns that have been discovered in the model results or from other factors. Gaining business understanding is an iterative process in data mining. The go or no-go decision must be made in this step to move to the deployment phase.

6. Deployment

The knowledge or information, which is gained through data mining process, needs to be presented in such a way that stakeholders can use it when they want it. Based on the business requirements, the deployment phase could be as simple as creating a report or as complex as a repeatable data mining process across the organization. In the deployment phase, the plans for deployment, maintenance, and monitoring have to be created for implementation and also future supports. From the project point of view, the final report of the project needs to summary the project experiences and review the project to see what need to improved created learned lessons.

These 6 steps describe the Cross-industry standard process for data mining, known as CRISP-DM. It is an open standard process model that describes common approaches used by data mining experts. It is the most widely-used analytics model.

Python | Pandas DataFrame

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

We will get a brief insight on all these basic operation which can be performed on Pandas DataFrame :

Creating a DataFrame

Dealing with Rows and Columns

Indexing and Selecting Data

Working with Missing Data

Iterating over rows and columns

Creating a Pandas DataFrame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc. Dataframe can be created in different ways here are some ways by which we create a dataframe:

Creating a dataframe using List: DataFrame can be created using a single list or a list of lists.

# import pandas as pd

import pandas as pd

# list of strings

lst = ['Geeks', 'For', 'Geeks', 'is',

'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list

df = pd.DataFrame(lst)

print(df)

Run on IDE

Output:

Creating DataFrame from dict of ndarray/lists: To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

# Python code demonstrate creating

# DataFrame from dict narray / lists

# By default addresses.

import pandas as pd

# intialise data of lists.

data = {'Name':['Tom', 'nick', 'krish', 'jack'],

'Age':[20, 21, 19, 18]}

# Create DataFrame

df = pd.DataFrame(data)

# Print the output.

print(df)

Run on IDE

Output:

For more details refer to Creating a Pandas DataFrame

Dealing with Rows and Columns

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.

Column Selection: In Order to select a column in Pandas DataFrame, we can either access the columns by calling them by their columns name.

# Import pandas package

import pandas as pd

# Define a dictionary containing employee data

data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],

'Age':[27, 24, 22, 32],

'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],

'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame

df = pd.DataFrame(data)

# select two columns

print(df[['Name', 'Qualification']])

Run on IDE

Output:

Row Selection: Pandas provide a unique method to retrieve rows from a Data frame. DataFrame.loc[] method is used to retrieve rows from Pandas DataFrame. Rows can also be selected by passing integer location to an iloc[] function.

Note: We’ll be using nba.csv file in below examples.

# importing pandas package

import pandas as pd

# making data frame from csv file

data = pd.read\_csv("nba.csv", index\_col ="Name")

# retrieving row by loc method

first = data.loc["Avery Bradley"]

second = data.loc["R.J. Hunter"]

print(first, "\n\n\n", second)

Output:

As shown in the output image, two series were returned since there was only one parameter both of the times.

For more Details refer to Dealing with Rows and Columns

Indexing and Selecting Data

Indexing in pandas means simply selecting particular rows and columns of data from a DataFrame. Indexing could mean selecting all the rows and some of the columns, some of the rows and all of the columns, or some of each of the rows and columns. Indexing can also be known as Subset Selection.

Indexing a Dataframe using indexing operator [] :

Indexing operator is used to refer to the square brackets following an object. The .loc and .iloc indexers also use the indexing operator to make selections. In this indexing operator to refer to df[].

Selecting a single columns

In order to select a single column, we simply put the name of the column in-between the brackets

# importing pandas package

import pandas as pd

# making data frame from csv file

data = pd.read\_csv("nba.csv", index\_col ="Name")

# retrieving columns by indexing operator

first = data["Age"]

print(first)

Output:

Indexing a DataFrame using .loc[ ] :

This function selects data by the label of the rows and columns. The df.loc indexer selects data in a different way than just the indexing operator. It can select subsets of rows or columns. It can also simultaneously select subsets of rows and columns.

Selecting a single row

In order to select a single row using .loc[], we put a single row label in a .loc function.

# importing pandas package

import pandas as pd

# making data frame from csv file

data = pd.read\_csv("nba.csv", index\_col ="Name")

# retrieving row by loc method

first = data.loc["Avery Bradley"]

second = data.loc["R.J. Hunter"]

print(first, "\n\n\n", second)

Output:

As shown in the output image, two series were returned since there was only one parameter both of the times.

Indexing a DataFrame using .iloc[ ] :

This function allows us to retrieve rows and columns by position. In order to do that, we’ll need to specify the positions of the rows that we want, and the positions of the columns that we want as well. The df.iloc indexer is very similar to df.loc but only uses integer locations to make its selections.

Selecting a single row

In order to select a single row using .iloc[], we can pass a single integer to .iloc[] function.

import pandas as pd

# making data frame from csv file

data = pd.read\_csv("nba.csv", index\_col ="Name")

# retrieving rows by iloc method

row2 = data.iloc[3]

print(row2)

Output:

For more Details refer

Indexing and Selecting Data with Pandas

Boolean Indexing in Pandas

Working with Missing Data

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in real life scenario. Missing Data can also refer to as NA(Not Available) values in pandas.

Checking for missing values using isnull() and notnull() :

In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.

# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from list

df = pd.DataFrame(dict)

# using isnull() function

df.isnull()

Output:

Filling missing values using fillna(), replace() and interpolate() :

In order to fill null values in a datasets, we use fillna(), replace() and interpolate() function these function replace NaN values with some value of their own. All these function help in filling a null values in datasets of a DataFrame. Interpolate() function is basically used to fill NA values in the dataframe but it uses various interpolation technique to fill the missing values rather than hard-coding the value.

# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)

# filling missing value using fillna()

df.fillna(0)

Output:

Dropping missing values using dropna() :

In order to drop a null values from a dataframe, we used dropna() function this fuction drop Rows/Columns of datasets with Null values in different ways.

# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

'Second Score': [30, np.nan, 45, 56],

'Third Score':[52, 40, 80, 98],

'Fourth Score':[np.nan, np.nan, np.nan, 65]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)

df

Now we drop rows with at least one Nan value (Null value)

# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

'Second Score': [30, np.nan, 45, 56],

'Third Score':[52, 40, 80, 98],

'Fourth Score':[np.nan, np.nan, np.nan, 65]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)

# using dropna() function

df.dropna()

Output:

For more Details refer to Working with Missing Data in Pandas

Iterating over rows and columns

Iteration is a general term for taking each item of something, one after another. Pandas DataFrame consists of rows and columns so, in order to iterate over dataframe, we have to iterate a dataframe like a dictionary.

Iterating over rows :

In order to iterate over rows, we can use three function iteritems(), iterrows(), itertuples() . These three function will help in iteration over rows.

# importing pandas as pd

import pandas as pd

# dictionary of lists

dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],

'degree': ["MBA", "BCA", "M.Tech", "MBA"],

'score':[90, 40, 80, 98]}

# creating a dataframe from a dictionary

df = pd.DataFrame(dict)

print(df)

Run on IDE

Now we apply iterrows() function in order to get a each element of rows.

# importing pandas as pd

import pandas as pd

# dictionary of lists

dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],

'degree': ["MBA", "BCA", "M.Tech", "MBA"],

'score':[90, 40, 80, 98]}

# creating a dataframe from a dictionary

df = pd.DataFrame(dict)

# iterating over rows using iterrows() function

for i, j in df.iterrows():

print(i, j)

print()

Run on IDE

Output:

Iterating over Columns :

In order to iterate over columns, we need to create a list of dataframe columns and then iterating through that list to pull out the dataframe columns.

# importing pandas as pd

import pandas as pd

# dictionary of lists

dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],

'degree': ["MBA", "BCA", "M.Tech", "MBA"],

'score':[90, 40, 80, 98]}

# creating a dataframe from a dictionary

df = pd.DataFrame(dict)

print(df)

Run on IDE

Now we iterate through columns in order to iterate through columns we first create a list of dataframe columns and then iterate through list.

# creating a list of dataframe columns

columns = list(df)

for i in columns:

# printing the third element of the column

print (df[i][2])

Output:

For more Details refer to Iterating over rows and columns in Pandas DataFrame

DataFrame Methods:

FUNCTION DESCRIPTION

index() Method returns index (row labels) of the DataFrame

insert() Method inserts a column into a DataFrame

add() Method returns addition of dataframe and other, element-wise (binary operator add)

sub() Method returns subtraction of dataframe and other, element-wise (binary operator sub)

mul() Method returns multiplication of dataframe and other, element-wise (binary operator mul)

div() Method returns floating division of dataframe and other, element-wise (binary operator truediv)

unique() Method extracts the unique values in the dataframe

nunique() Method returns count of the unique values in the dataframe

value\_counts() Method counts the number of times each unique value occurs within the Series

columns() Method returns the column labels of the DataFrame

axes() Method returns a list representing the axes of the DataFrame

isnull() Method creates a Boolean Series for extracting rows with null values

notnull() Method creates a Boolean Series for extracting rows with non-null values

between() Method extracts rows where a column value falls in between a predefined range

isin() Method extracts rows from a DataFrame where a column value exists in a predefined collection

dtypes() Method returns a Series with the data type of each column. The result’s index is the original DataFrame’s columns

astype() Method converts the data types in a Series

values() Method returns a Numpy representation of the DataFrame i.e. only the values in the DataFrame will be returned, the axes labels will be removed

sort\_values()- Set1, Set2 Method sorts a data frame in Ascending or Descending order of passed Column

sort\_index() Method sorts the values in a DataFrame based on their index positions or labels instead of their values but sometimes a data frame is made out of two or more data frames and hence later index can be changed using this method

loc[] Method retrieves rows based on index label

iloc[] Method retrieves rows based on index position

ix[] Method retrieves DataFrame rows based on either index label or index position. This method combines the best features of the .loc[] and .iloc[] methods

rename() Method is called on a DataFrame to change the names of the index labels or column names

columns() Method is an alternative attribute to change the coloumn name

drop() Method is used to delete rows or columns from a DataFrame

pop() Method is used to delete rows or columns from a DataFrame

sample() Method pulls out a random sample of rows or columns from a DataFrame

nsmallest() Method pulls out the rows with the smallest values in a column

nlargest() Method pulls out the rows with the largest values in a column

shape() Method returns a tuple representing the dimensionality of the DataFrame

ndim() Method returns an ‘int’ representing the number of axes / array dimensions.

Returns 1 if Series, otherwise returns 2 if DataFrame

dropna() Method allows the user to analyze and drop Rows/Columns with Null values in different ways

fillna() Method manages and let the user replace NaN values with some value of their own

rank() Values in a Series can be ranked in order with this method

query() Method is an alternate string-based syntax for extracting a subset from a DataFrame

copy() Method creates an independent copy of a pandas object

duplicated() Method creates a Boolean Series and uses it to extract rows that have duplicate values

drop\_duplicates() Method is an alternative option to identifying duplicate rows and removing them through filtering

set\_index() Method sets the DataFrame index (row labels) using one or more existing columns

reset\_index() Method resets index of a Data Frame. This method sets a list of integer ranging from 0 to length of data as index

where() Method is used to check a Data Frame for one or more condition and return the result accordingly. By default, the rows not satisfying the condition are filled with NaN value