Module 3

Data Analytics with Python - Applied analytics

Section: Analyzing data with python

Pandas

Pandas is an open-source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Pandas is built on top of the NumPy package; hence it takes a lot of basic inspiration from it.

The Data Structures provided by Pandas are of two distinct types

1. Pandas Series
2. Pandas DataFrame

Series
A Series is a one-dimensional object that can hold any data type such as integers, floats and strings. Let's take a list of items as an input argument and create a Series object for that list.

```
import pandas as pd #importing pandas library

series = pd.Series([5,4,8,9,25,6]) # creating series - its one dimensional data

series
```

Output-

```
0      5
1      4
2      8
3      9
4     25
5      6
dtype: int64
```

Series also generates by default row index numbers which is a sequence of incremental numbers starting from '0'

Dataframe

A DataFrame is a two-dimensional object that can have columns with potential different types. Different kind of inputs include dictionaries, lists, series, and even another DataFrame.

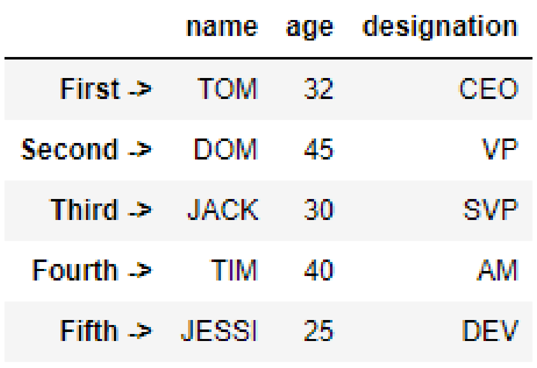
It is the most commonly used pandas object

Let’s create dataframe

Example-1

```
dataframe_dict = {

'name' : ["TOM", "DOM", "JACK", "TIM", "JESSI"],

'age' : [32,45, 30, 40, 25],

'designation': ["CEO", "VP", "SVP", "AM", "DEV"]

}

df = pd.DataFrame( dataframe_dict,

index = [ "First -> ","Second -> ", "Third -> ", "Fourth -> ",  "Fifth -> "])
```

Output-



	name	age	designation
First ->	TOM	32	CEO
Second ->	DOM	45	VP
Third ->	JACK	30	SVP
Fourth ->	TIM	40	AM
Fifth ->	JESSI	25	DEV

Properties of A DataFrame:

- The **axes** attribute of DataFrame contains both the row axis index and the column axis index.
- The shape attribute has the shape of the 2-dimensional matrix/DataFrame as a tuple. e.g., A shape of (2,1) means a DataFrame instance with 2 rows and 1 column, a shape (5,3) means a DataFrame instance with 5 rows and 3 columns. As shown in above example-1.

- A DataFrame has several columns and rows to form its structure.
- Each column can hold different types of elements. One column of a DataFrame can hold all integers while another column has all its elements as string.
- A column itself can hold objects of several types. It is possible for a DataFrame column to have some of its members as integers and some as floats and the remaining elements as an instance of a class like complex.
- A column with heterogeneous data elements will have its type as Object.
- The Python above example-1 has a DataFrame with first column as object , second column as all integers and the last column with objects again.
- The DataFrame attribute dtypes returns type of each column contained in a DataFrame series.

NumPy

NumPy (**Numerical Python**) is an open-source Python library that's used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python ecosystems. NumPy users include everyone from beginning coders to experienced researchers doing state-of-the-art scientific and industrial research and development. The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.

- It is a powerful N-dimensional array object.
- Sophisticated (broadcasting/universal) functions.
- Useful linear algebra, Fourier transform, and random number capabilities.
- NumPy can also be used as an efficient multi-dimensional container of generic data.

Example-2

```
import numpy as np

civilian_birth = np.array([4352, 233, 3245, 256, 2394])

civilian_birth

#output

array([4352,  233, 3245,  256, 2394])
```

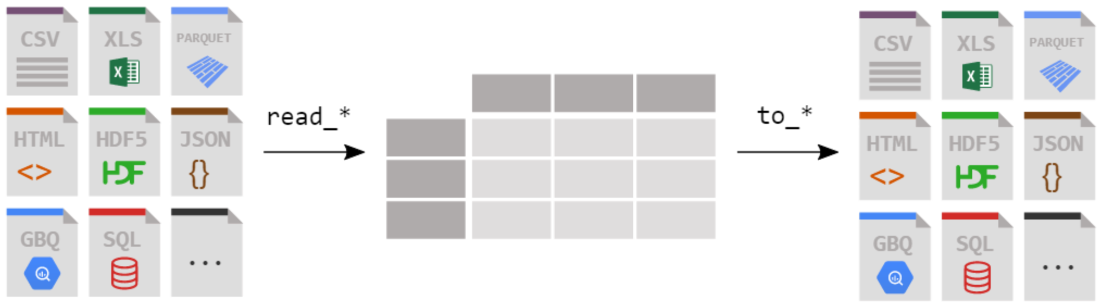
NumPy datatypes

NumPy supports a wider variety of data types than are built-in to the Python language by default. They are defined by the NumPy. dtype class and include:

- intc (same as a C integer) and intp (used for indexing)
- int8, int16, int32, int64
- uint8, uint16, uint32, uint64
- float16, float32, float64
- complex64, complex128
- bool_, int_, float_, complex_ are shorthand for defaults.

These can be used as functions to cast literals or sequence types, as well as arguments.
to numpy functions that accept the dtype keyword argument.

Importing excel sheets, csv files, executing sql queries



The pandas I/O API is a set of top-level reader functions to read and write different type of files we use this read function of pandas and store it in new dataframe. After importing desired file, we can run desired operation on that dataframe.

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	LaTeX		Styler.to_latex
text	XML	read_xml	to_xml
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	

binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

Examples of Importing different file formats using pandas

Import CSV File into Pandas**.**

Function syntex

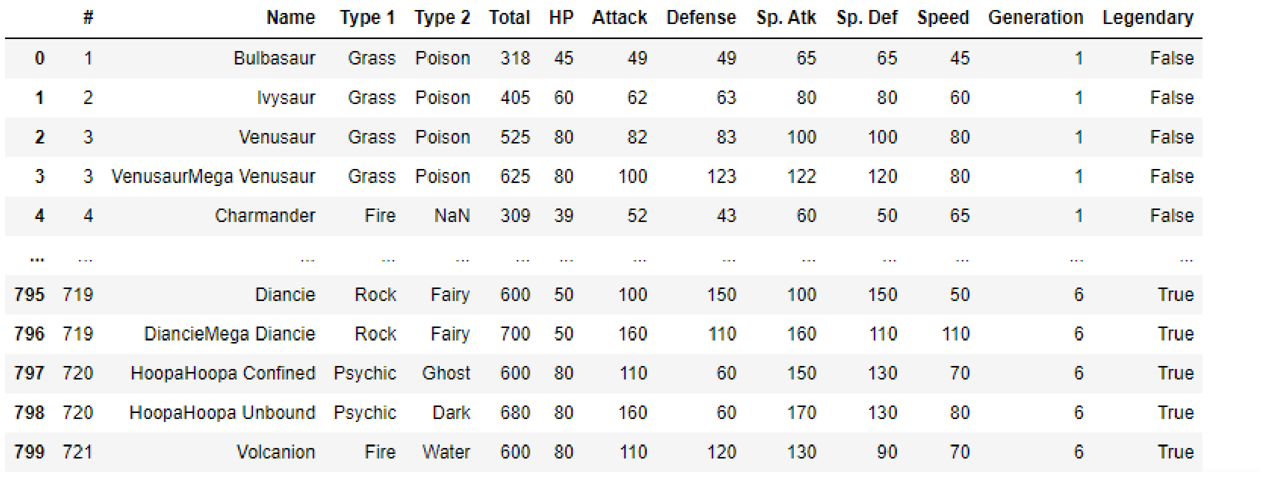
pd. read_csv (r‘Path where file is stored\Filename.csv’)

Place "r" before the path string to address special character, such as '\'.

Example-3

```
import pandas as pd
path = 'Pokemon.csv'    # define complete path of file location
# Load the csv file into a data frame
df = pd.read_csv(path)
df
```

Output



	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
795	719	Diancie	Rock	Fairy	600	50	100	150	100	150	50	6	True
796	719	DiancieMega Diancie	Rock	Fairy	700	50	160	110	160	110	110	6	True
797	720	HoopaHoopa Confined	Psychic	Ghost	600	80	110	60	150	130	70	6	True
798	720	HoopaHoopa Unbound	Psychic	Dark	680	80	160	60	170	130	80	6	True
799	721	Volcanion	Fire	Water	600	80	110	120	130	90	70	6	True

Exporting CSV File into Pandas.

To export dataframe to desired format we can use below '.to_' function of pandas.

Example-4

```
df.to_csv('Pokemon.csv')
```

Now data frame is saved as csv

Similarly, we can use we can use below Import/Export functions for different type of files.

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	LaTeX		Styler.to_latex
text	XML	read_xml	to_xml
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet

binary	ORC Format	read_orc	
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

Executing SQL queries

Example-5

```
import sqlite3

# create a database or connect to existing database
conn = sqlite3.connect("mydb.db")
# create a table

query = "create table if not exists customers (cid int primary key not null, Name text
not null, Age int not null, City text not null);"
conn.execute(query)
# inserting data into the table

query = "insert into customers(cid,Name,Age,City) values(?,?,?,?)"
conn.execute(query,(101,"Anshu",45,"Delhi"))
conn.execute(query,(102,"Max",37,"Chennai"))
conn.execute(query,(103,"Jenny",25,"Mumbai"))


# select query = to fetch data

query = "select * from customers"
data= conn.execute(query)
for row in data:
    print(row)
```

Output

```
(101, 'Anshu', 45, 'Delhi')

(102, 'Max', 37, 'Chennai')

(103, 'Jenny', 25, 'Mumbai')
```

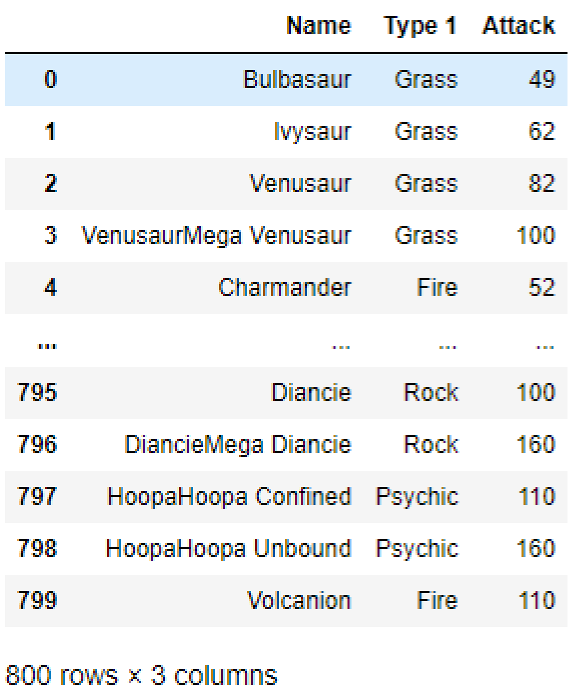
Selection of columns and Filtering Dataframes

We can select a column from a dataframe by using the column name we want to select

Example-6

```
df[['Name','Type 1','Attack']]  #select only Name,Type 1,Attack columns
```

Output



	Name	Type 1	Attack
0	Bulbasaur	Grass	49
1	Ivysaur	Grass	62
2	Venusaur	Grass	82
3	VenusaurMega Venusaur	Grass	100
4	Charmander	Fire	52
...	...	...	...
795	Diancie	Rock	100
796	DiancieMega Diancie	Rock	160
797	HoopaHoopa Confined	Psychic	110
798	HoopaHoopa Unbound	Psychic	160
799	Volcanion	Fire	110

800 rows × 3 columns

Filtering of data

Data filtering is another powerful feature of Pandas

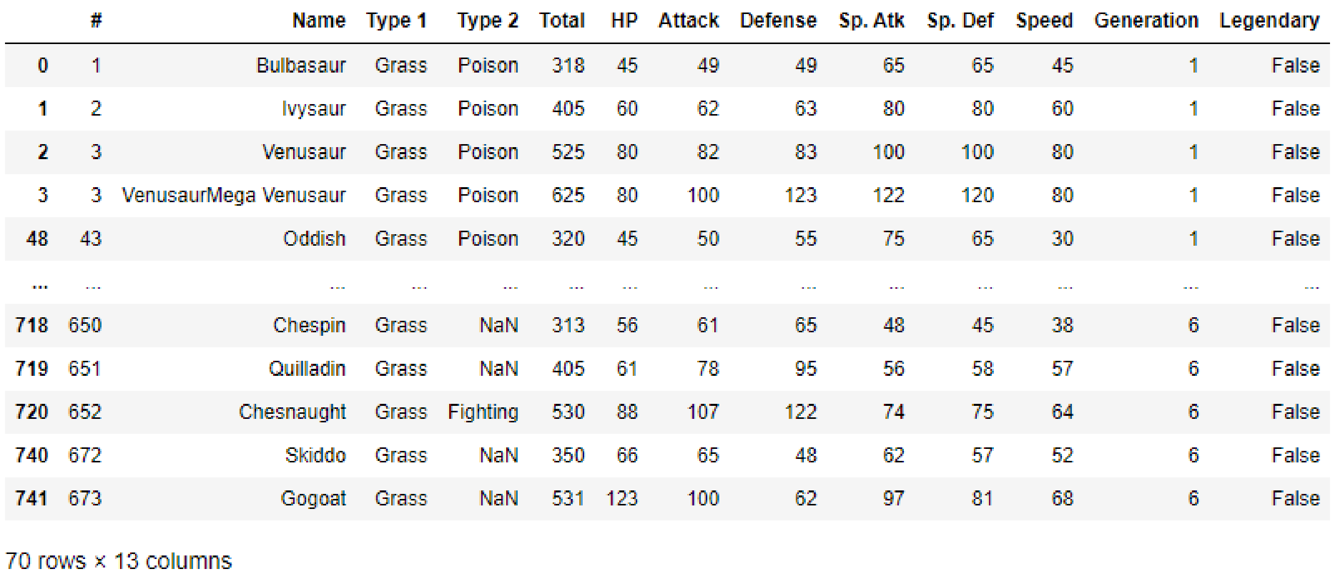
You can create very powerful and sophisticated expressions by combining logical operations with the following operators:

- **NOT** (~)
- **AND** (&)
- **OR** (|)
- **XOR** (^)

Example-7

```
new_df = df[df['Type 1'] == 'Grass'] # Filter rows containing Type 1 = Grass type (filtering grass type pokemon)

new_df
```

Output



	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
48	43	Oddish	Grass	Poison	320	45	50	55	75	65	30	1	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
718	650	Chespin	Grass	NaN	313	56	61	65	48	45	38	6	False
719	651	Quilladin	Grass	NaN	405	61	78	95	56	58	57	6	False
720	652	Chesnaught	Grass	Fighting	530	88	107	122	74	75	64	6	False
740	672	Skiddo	Grass	NaN	350	66	65	48	62	57	52	6	False
741	673	Gogoat	Grass	NaN	531	123	100	62	97	81	68	6	False

70 rows × 13 columns

As you can see, we have 70 rows filtered by Type 1 = Grass which suggest we have 70 pokemon having Type 1 as grass in whole data set.

Descriptive Analysis with pandas

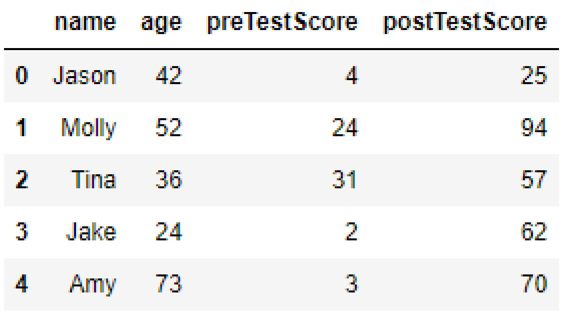
Python Pandas is used to compute descriptive statistical data like **count, unique values, mean, standard deviation, minimum and maximum value** and many more. In this section let's learn to get the descriptive statistics for Pandas DataFrame.

Let's create simple dataframe

Example-8

```
import pandas as pd

data = {'name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],

        'age': [42, 52, 36, 24, 73],

        'preTestScore': [4, 24, 31, 2, 3],

        'postTestScore': [25, 94, 57, 62, 70]}

df = pd.DataFrame(data, columns = ['name', 'age', 'preTestScore', 'postTestScore'])
```

Output



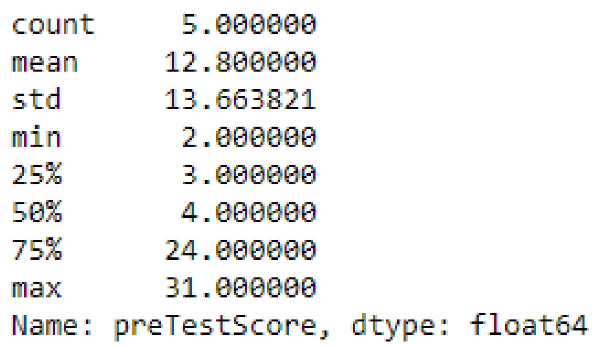
	name	age	preTestScore	postTestScore
0	Jason	42	4	25
1	Molly	52	24	94
2	Tina	36	31	57
3	Jake	24	2	62
4	Amy	73	3	70

We can use. Describe () function to compute descriptive statistical data of desired column.

Example-9

```
df['preTestScore'].describe() # describing preTestScore column
```

Output



```
count     5.000000
mean     12.800000
std      13.663821
min       2.000000
25%       3.000000
50%       4.000000
75%      24.000000
max      31.000000
Name: preTestScore, dtype: float64
```

As shown above we can se there is 5 nos of row from count, mean of preTestScore data is 12 minimum values of score are 2 and maximum score is 31. like this we can get statistical information about data which can help us to further evaluate the results.

Other examples of functions used for descriptive analysis are as below.

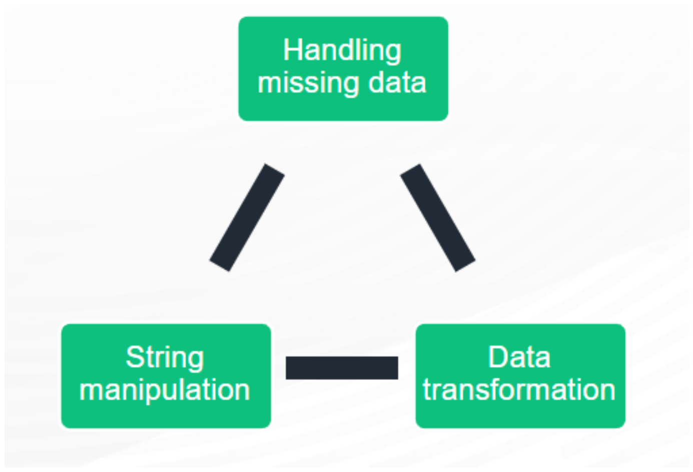
```
# The sum of all the ages
df['age'].sum()

# Mean of preTestScore
df['preTestScore'].mean()

# Cumulative sum of preTestScores, moving from the rows
df['preTestScore'].cumsum()

# Count the number of non-NA values
df['preTestScore'].count()

# Minimum value of preTestScore
df['preTestScore'].min()

# Maximum value of preTestScore
df['preTestScore'].max()

# Median value of preTestScore
df['preTestScore'].median()

# Sample variance of preTestScore values
df['preTestScore'].var()

# Sample standard deviation of preTestScore values
df['preTestScore'].std()

# Correlation Matrix Of  Values
df.corr()

# Covariance Matrix Of Values
df.cov()

# Skewness of preTestScore values
df['preTestScore'].skew()

# Kurtosis of preTestScore values
df['preTestScore'].kurt()
```

Data Cleaning and Preparation

Data preparation is one of the most important steps in data analysis and modeling.

- A significant amount of time, i.e., close to 80% of the time, is spent in data preparation processes like loading, cleaning, transforming, and rearranging.
- Pandas, along with many Pythons language features, provides a high-level, flexible, and fast set of tools for manipulating data in the right form.

The most important data cleaning and preparation methods are as follows



Handling Missing Data

- Missing data normally occurs in many data analysis applications.
- The way of representation of missing data in pandas' objects is a little imperfect, but it is functional for a lot of users.
- For numeric data, pandas use the floating-point value NaN (Not a Number) to represent missing data, which is called sentinel value.
- Pandas uses a convention used in the R programming language by referring to missing data as NA, which stands for not available.
- During data cleaning, it is very important to do the analysis on the missing data itself to identify data collection problems or potential biases in the data caused by missing data.
- The two common techniques of handling missing data are as follows:
    - Filtering out missing data
    - Filling in missing data

Filtering Out Missing Data

- There are different ways to filter out missing data using Pandas. The most obvious way is to do manually using pandas.isnull and boolean indexing.
- dropna is also helpful in filtering out missing data. When used on a Series, it returns the Series with only the non-null data and index values.
- With DataFrame objects, we will drop rows or columns that are all NA or only those containing any NAs. dropna by default drops any row containing a missing value.
- Another way of filtering out DataFrame rows tends to concern time series data. In case we need to keep only rows containing a certain number of observations. This can be indicated with the thresh argument.

Filling in Missing Data

- Another way of handling missing data is that filling the gaps in any number of ways.
- In most of the cases, the fillna method is used, calling which with a constant replaces missing values with that value.
- Calling fillna with a dict, different fill value can be used for each column.
- fillna returns a new object, but the existing object in-place can be modified.
- The same interpolation methods available for reindexing can be used with fillna.

Removing Duplicates

- Calling the method duplicated returns a boolean Series, which indicates whether each row is a duplicate (has been observed in a previous row) or not.
- The drop_duplicates method returns a DataFrame where the duplicated array is False.
- Both the duplicated and drop_duplicates methods consider all the columns, by default; but we can specify any subset of columns to detect duplicates.
- duplicated and drop_duplicates methods by default keep the first observed value combination. Passing the argument keep='last' will return the last one.

Examples-10

```
# Drop missing observations
df_no_missing = df.dropna()

# Drop rows where all cells in that row is NA
df_cleaned = df.dropna(how='all’)

# Create a new column full of missing values
df['location'] = np.nan

# Drop column if they only contain missing values
df.dropna(axis=1, how='all’)

# Drop rows that contain less than five observations
# This is really mostly useful for time series
df.dropna(thresh=5)

# Fill in missing data with zeros
df.fillna(0)

# inplace=True means that the changes are saved to the df right away
df["preTestScore"].fillna(df["preTestScore"].mean(), inplace=True)

# Fill in missing in postTestScore with each sex’s mean value
of postTestScore
df["postTestScore"].fillna(df.groupby("sex")["postTestScore"].transform
("mean"), inplace=True)


# Select some rows but ignore the missing data points
# Select the rows of df where age is not NaN and sex is not NaN
df[df['age'].notnull() & df['sex'].notnull()]
```

Example of Deleting Duplicate values

```
import pandas as pd
# Create dataframe with duplicates
raw_data = {'first_name': ['Jason', 'Jason', 'Jason','Tina', 'Jake',
'Amy'],
        'last_name': ['Miller', 'Miller', 'Miller','Ali', 'Milner',
'Cooze'],
        'age': [42, 42, 1111111, 36, 24, 73],
        'preTestScore': [4, 4, 4, 31, 2, 3],
        'postTestScore': [25, 25, 25, 57, 62, 70]}

df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age',
'preTestScore', 'postTestScore’])

# Identify which observations are duplicates
df.duplicated()
df.drop_duplicates()

# Drop duplicates in the  name column, but take the last obs in the
duplicated set
df.drop_duplicates(['first_name'], keep='last')
```

Analyzing Outliers

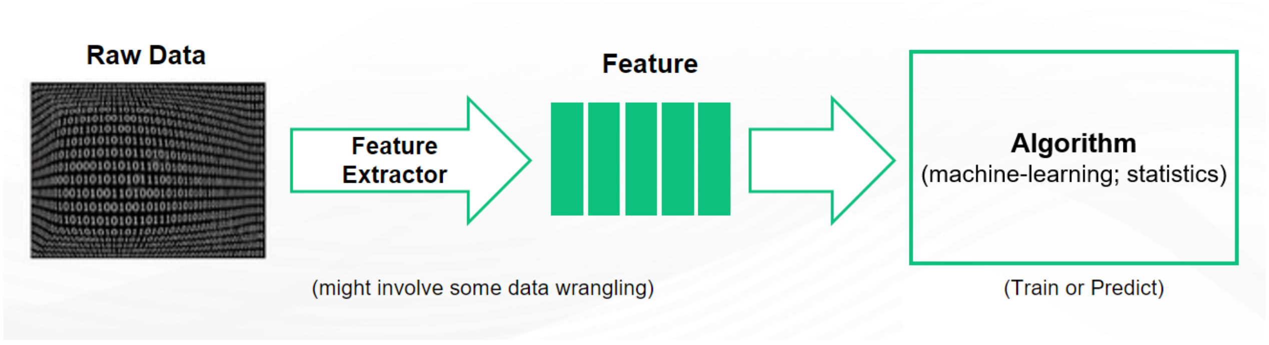
- Trimming: It excludes the outlier values from our analysis. By applying this technique our data becomes thin when there are more outliers present in the dataset. Its main advantage is its fastest nature.
- Capping: In this technique, we cap our outliers data and make the limit i.e, above a particular value or less than that value, all the values will be considered as outliers, and the number of outliers in the dataset gives that capping number.

  For Example, if you’re working on the income feature, you might find that people above a certain income level behave in the same way as those with a lower income. In this case, you can cap the income value at a level that keeps that intact and accordingly treat the outliers.

- Treat outliers as a missing value: By assuming outliers as the missing observations, treat them accordingly i.e, same as those of missing values.
- Discretization: In this technique, by making the groups we include the outliers in a particular group and force them to behave in the same manner as those of other points in that group. This technique is also known as Binning.

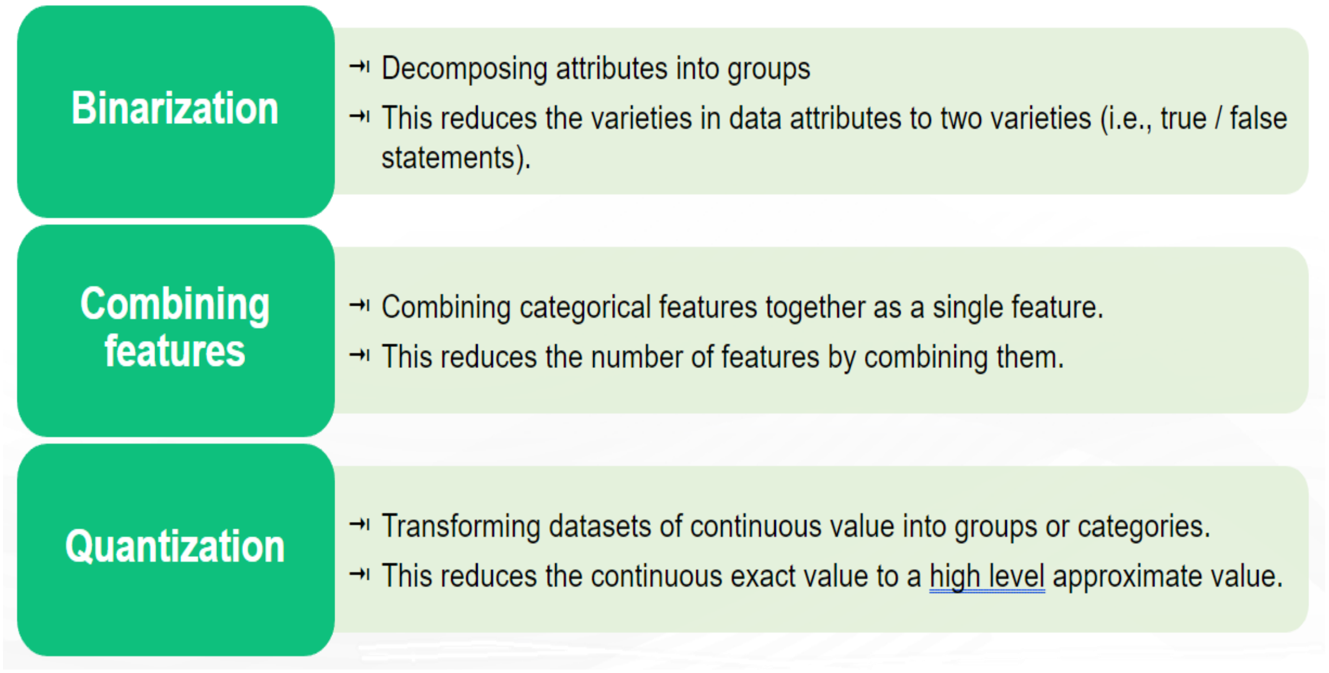
Feature Engineering

- In machine learning, transforming raw data to an optimal set of features is important to increase the accuracy of the results. This process is known as feature engineering.
- More the number of appropriate features, the greater the ML model can perform. For this, feature engineering enforces the data to be correctly represented.
- This better representation of the problem to predictive models results in high accuracy of the model.


Raw Data
Feature Extractor
Feature
Algorithm
(machine-learning; statistics)
(might involve some data wrangling)
(Train or Predict)

- Feature engineering is the most changing part in creating a model that yields highly accurate results.
- Since ML libraries take care of implementations, this process consumes more than three-fourth of the time.
- Deciding these features are specific to the domain and so it requires domain expertise, practice and many trails & errors.
- Generally, this process is iterative. It requires applying a technique, evaluating a model and checking if the accuracy has improved.

Techniques in Feature Engineering



Binarization

- Decomposing attributes into groups
- This reduces the varieties in data attributes to two varieties (i.e., true / false statements).

Combining features

- Combining categorical features together as a single feature.
- This reduces the number of features by combining them.

Quantization

- Transforming datasets of continuous value into groups or categories.
- This reduces the continuous exact value to a high level approximate value.

Example for Feature Engineering

Language Prediction

Feature engineering is used in language prediction. Consider we’re given some statements in a language and we need to predict the language.

```
English:
What’s your name?
How’s the weather?
What’s your favorite movie?
```

```
French:
Comment vous appelez-vous?
Quel temps fait-il?
Quel est ton/votre film préféré?
```

Group by operations

groupby() function is used to split the data into groups based on some criteria. pandas' objects can be split on any of their axes. The abstract definition of grouping is to provide a mapping of labels to group names.

Grouping records by column(s) is a common need for data analyses. Such scenarios include counting employees in each department of a company, calculating the average salary of male and female employees respectively in each department, and calculating the average salary of employees of different ages. Pandas has groupby function to be able to handle most of the grouping tasks conveniently. But there are certain tasks that the function finds it hard to manage. Here let's examine these "difficult" tasks and try to give alternative solutions.

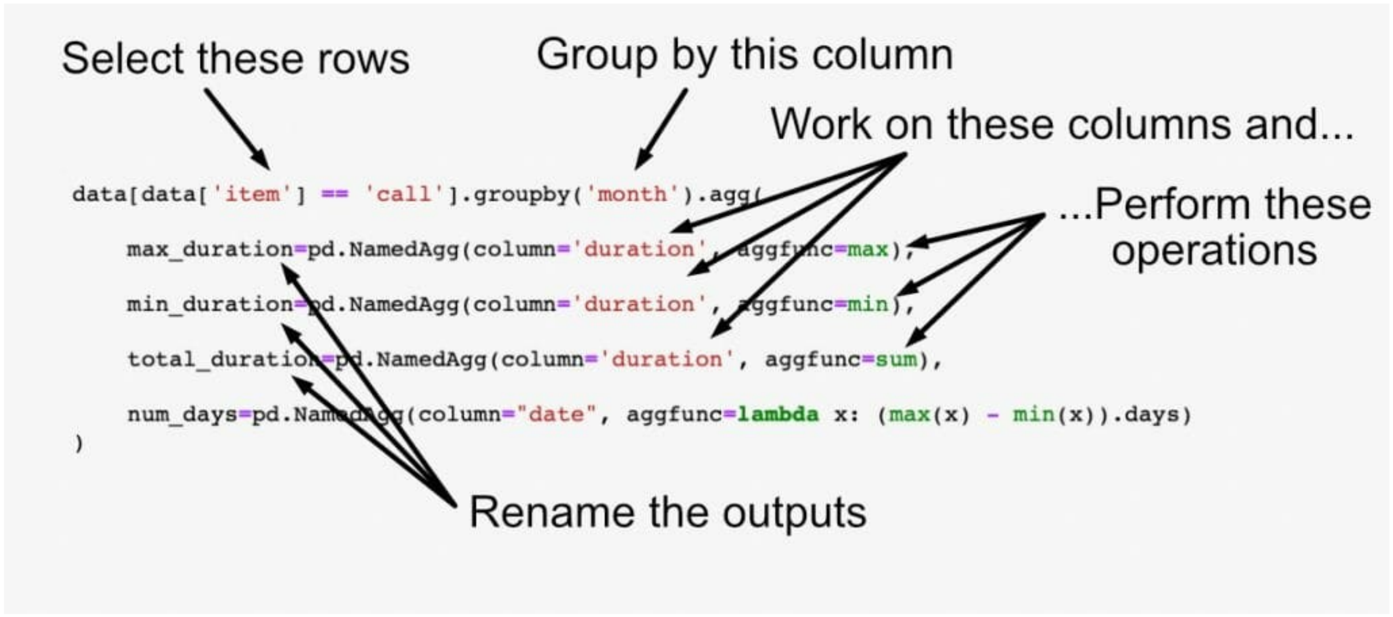
groupby() is one of the most important Pandas functions. It is used to group and summarize records according to the split-apply-combine strategy.

Groupby Statistical Analysis

The aggregation functionality provided by the agg() function allows multiple statistics to be calculated per group in one calculation.

Instructions for aggregation are provided in the form of a python dictionary or list. The dictionary keys are used to specify the columns upon which you'd like to perform operations, and the dictionary values to specify the function to run.

Refer below example for better understanding.



```
data[data['item'] == 'call'].groupby('month').agg(
    max_duration=pd.NamedAgg(column='duration', aggfunc=max),
    min_duration=pd.NamedAgg(column='duration', aggfunc=min),
    total_duration=pd.NamedAgg(column='duration', aggfunc=sum),
    num_days=pd.NamedAgg(column="date", aggfunc=lambda x: (max(x) - min(x)).days)
)
```

Summary

- Pandas has two types of data structure series and data frames
- A DataFrame is a two-dimensional object that can have columns with potential different types.
- The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.
- Pandas can be used to work with different type of files like-Excel, CSV,Tex,tSQL,Web data, Image.
- Python Pandas is used to compute descriptive statistical data like count, unique values, mean, standard deviation, minimum and maximum value and many more.
- Data frames can be filtered using You can create very powerful and sophisticated expressions by combining logical operations like- NOT (~), AND (&), OR (|), XOR (^).
- The two common techniques of handling missing data are - Filtering out missing data & filling in missing data.
- In machine learning, transforming raw data to an optimal set of features is important to increase the accuracy of the results. This process is known as feature engineering.
- groupby() function is used to split the data into groups based on some criteria. pandas' objects can be split on any of their axes. The abstract definition of grouping is to provide a mapping of labels to group names.