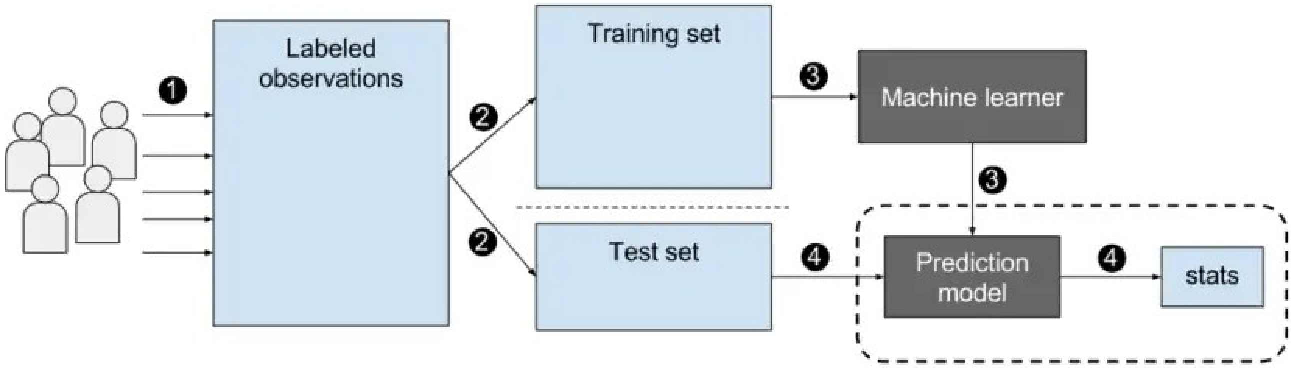
Assessment Questions

Batch: BD-DS-(ML) September batch
Faculty: Christy
Student Name: Indhulekha K J

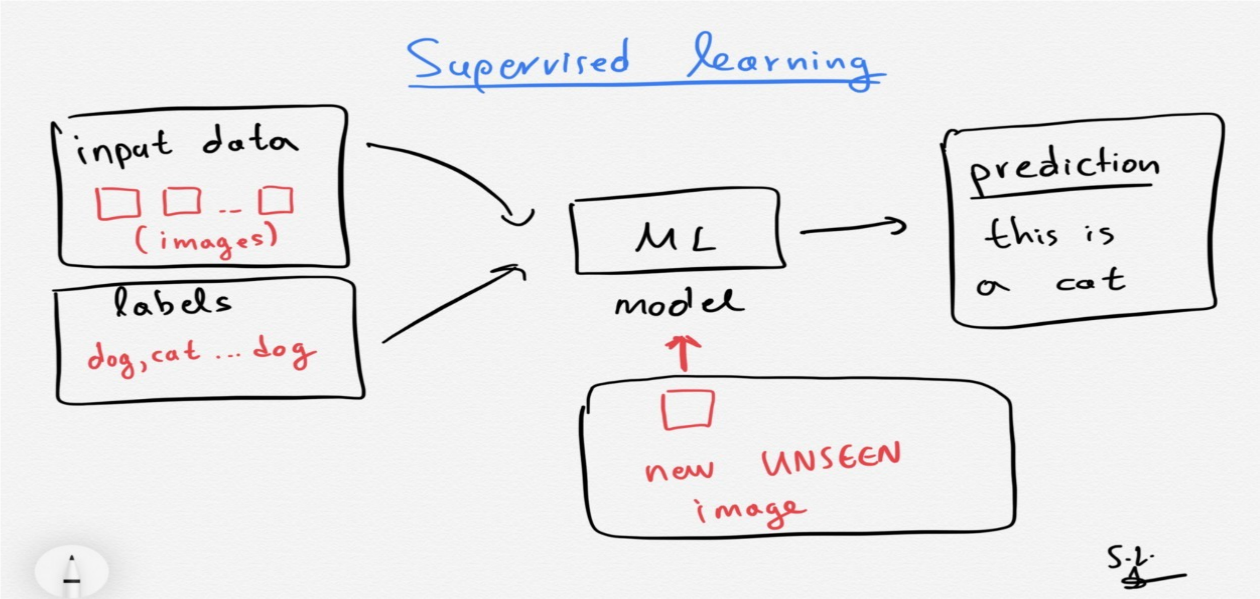
Q1. What is ML? What are the differences between supervised, unsupervised and reinforcement learning?
Machine learning is a sub-field of artificial intelligence (AI) that provides **systems** the ability to **automatically learn** and **improve** from **experience without** being **explicitly** programmed.
The main machine learning categories are,

1. Supervised machine learning

These models learn from the labeled dataset and then are used to predict future events. For the training procedure, the input is a known training data set with its corresponding labels, and the learning algorithm produces an inferred function to finally make predictions about some new unseen observations that one can give to the model. The model is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct intended output.



For example, the observations could be images of animals and the labels the name of the animal (e.g. cat, dog etc).



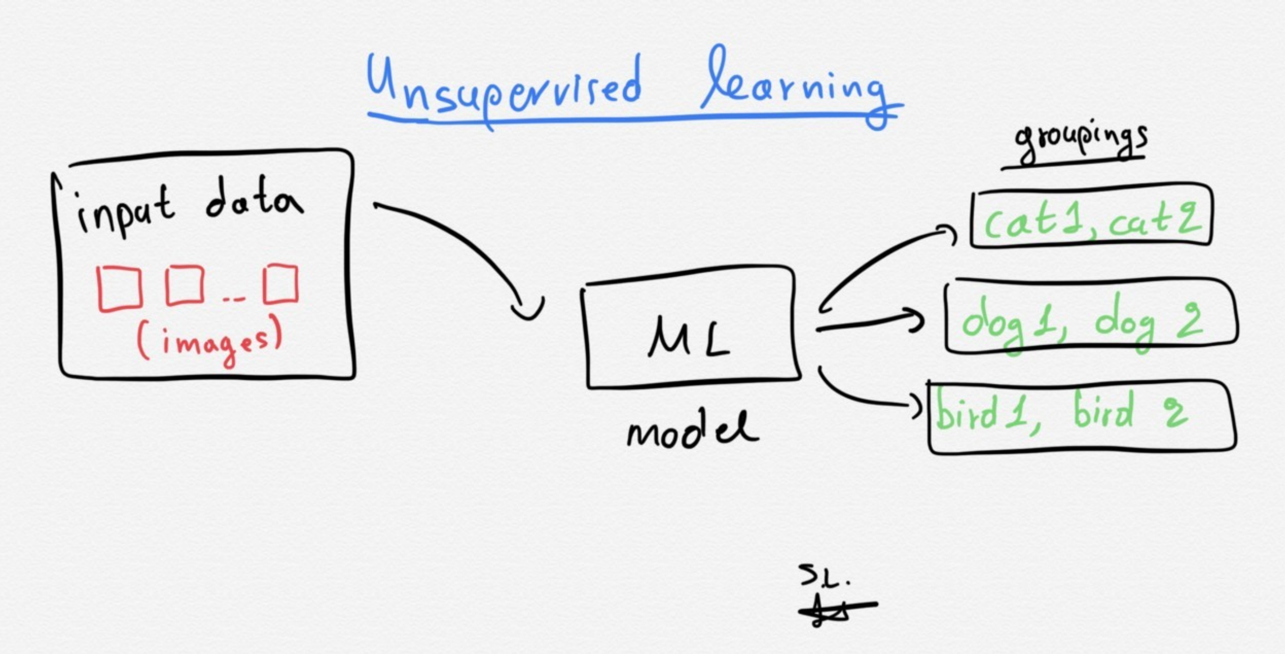
Supervised models can be further grouped into **regression** and classification **cases**:

- **Classification**: A classification problem is when the output variable is a category e.g. "disease" / "no disease".
- **Regression**: A regression problem is when the output variable is a real continuous value e.g. stock price prediction

Examples of models that belong to this family are the following: SVC, LDA, SVR, regression, random forests, Decision trees, KNN, Naive Bayes, XGBoost.

2. Unsupervised machine learning

Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't predict the right output, but instead, it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.



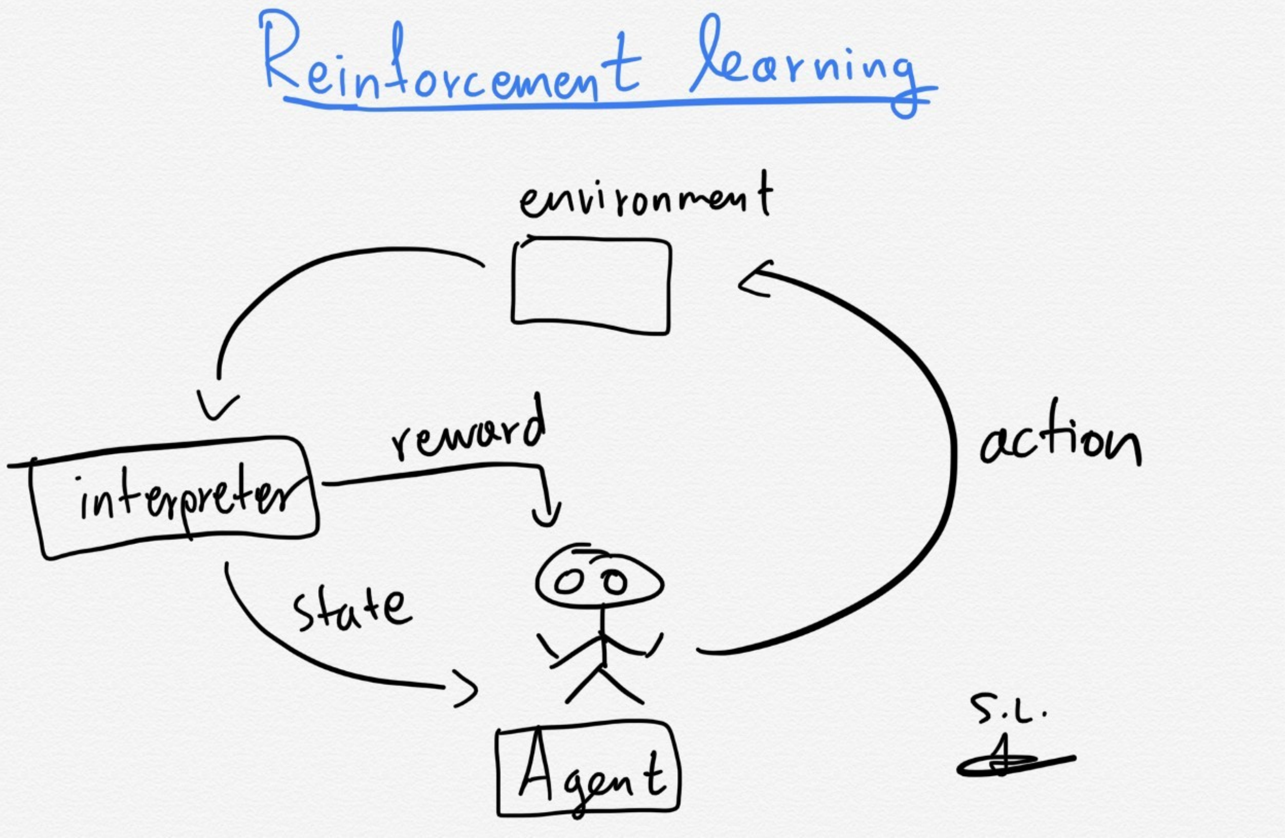
For this family of models, the research needs to have at hand a dataset with some observations without the need of having also the labels/classes of the observations.

Unsupervised models can be further grouped into **clustering** and **association** cases.

Clustering: A clustering problem is where you want to unveil the **inherent groupings** in the data, such as grouping animals based on some characteristics/features e.g. number of legs.

3. Reinforcement machine learning

This family of models consists of algorithms that use the estimated errors as rewards or penalties. If the error is big, then the penalty is high and the reward low. If the error is small, then the penalty is low and the reward high.



Trial error search and **delayed reward** are the most relevant characteristics of reinforcement learning. This family of models allows the automatic determination of the ideal behavior within a specific context in order to maximize the desired performance.

Reward feedback is required for the model to learn which action is best and this is known as "**the reinforcement signal**".

Comparison Table

Criteria	**Supervised ML**	**Unsupervised ML**	**Reinforcement ML**
Definition	Learns by using labelled data	Trained using unlabelled data without any guidance.	Works on interacting with the environment
Type of data	Labelled data	Unlabelled data	No – predefined data
Type of problems	Regression and classification	Association and Clustering	Exploitation or Exploration
Supervision	Extra supervision	No supervision	No supervision
Algorithms	Linear Regression, Logistic Regression, SVM, KNN etc.	K – Means, C – Means, Apriori	Q – Learning, SARSA
Aim	Calculate outcomes	Discover underlying patterns	Learn a series of action
Application	Risk Evaluation, Forecast Sales	Recommendation System, Anomaly Detection	Self Driving Cars, Gaming, Healthcare

Q2. List out the various classification algorithms and explain briefly?

Classification algorithms in machine learning use input training data to predict the likelihood that subsequent data will fall into one of the predetermined categories. One of the most common uses of classification is filtering emails into “spam” or “non-spam.”

Types of Classification Algorithms

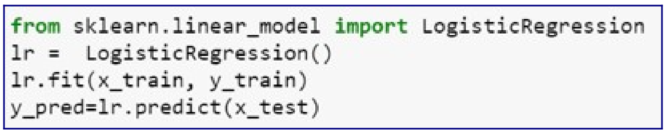
1. Logistic Regression
2. Naïve Bayes
3. XGBoost classifier
4. K-Nearest Neighbours
5. Decision Tree
6. Random Forest
7. Support Vector Machine

1. Logistic Regression

Definition: Logistic regression is a machine learning algorithm for classification. In this algorithm, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.

Advantages: Logistic regression is designed for this purpose (classification), and is most useful for understanding the influence of several independent variables on a single outcome variable.

Disadvantages: Works only when the predicted variable is binary, assumes all predictors are independent of each other and assumes data is free of missing values.



```
from sklearn.linear_model import LogisticRegression
lr =  LogisticRegression()
lr.fit(x_train, y_train)
y_pred=lr.predict(x_test)
```

LR is a transformation of a linear regression using the sigmoid function.

Sigmoid Function, f(x) = 1/(1+e^-x)

The distribution of Sigmoid function will be a S-curved function.

In General,
If f(x)>=0.5 Dependent Variable/Target Variable(DV) =1
If f(x)<0.5 Dependent Variable/Target Variable(DV) =0

2. Naïve Bayes

Definition: Naive Bayes classifiers are a collection of classification algorithms based on Bayes’ Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

The fundamental Naive Bayes assumption is that each feature makes an independent and equal contribution to the outcome.We assume that no pair of features are dependent. Secondly, each feature is given the same weight(or importance). None of the attributes is irrelevant and assumed to be contributing equally to the outcome.

Bayes’ Theorem finds the probability of an event occurring given the probability of another event that has already occurred

Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence.

P(A) is the priori of A (the prior probability, i.e. Probability of event before evidence is seen).

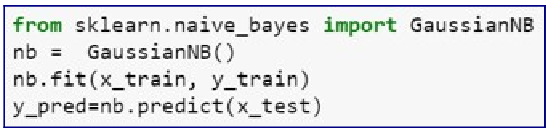
The evidence is an attribute value of an unknown instance(here, it is event B).

P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

$$P(A|B) - P(A)*P(B|A)/P(B)$$

Advantages: This algorithm requires a small amount of training data to estimate the necessary parameters. Naive Bayes classifiers are extremely fast compared to more sophisticated methods.

Disadvantages: Naive Bayes is known to be a bad estimator.



```
from sklearn.naive_bayes import GaussianNB
nb =  GaussianNB()
nb.fit(x_train, y_train)
y_pred=nb.predict(x_test)
```

3. XGBoost

XGBoost or extreme gradient boosting is one of the well-known gradient boosting techniques(ensemble) having enhanced performance and speed in tree-based (sequential decision trees) machine learning algorithms.

Features of XGBoost:

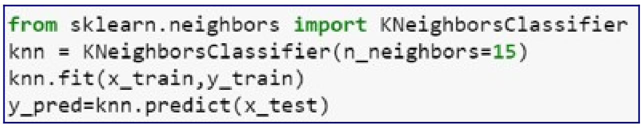
1. Can be run on both single and distributed systems(Hadoop, Spark).
2. XGBoost is used in supervised learning(regression and classification problems).
3. Supports parallel processing.
4. Cache optimization.
5. Efficient memory management for large datasets exceeding RAM.
6. Has a variety of regularizations which helps in reducing overfitting.
7. Auto tree pruning – Decision tree will not grow further after certain limits internally.
8. Can handle missing values.
9. Has inbuilt Cross-Validation.
10. Takes care of outliers to some extent.

4. K-Nearest Neighbours

Definition: Neighbours based classification is a type of lazy learning as it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the k nearest neighbours of each point.

Advantages: This algorithm is simple to implement, robust to noisy training data, and effective if training data is large.

Disadvantages: Need to determine the value of K and the computation cost is high as it needs to compute the distance of each instance to all the training samples.



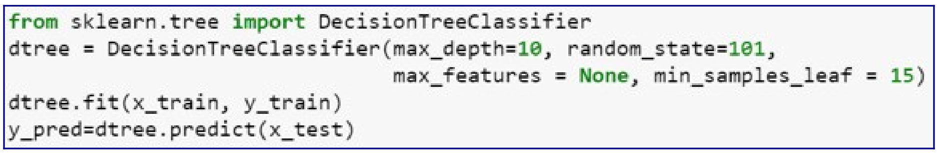
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(x_train,y_train)
y_pred=knn.predict(x_test)
```

5. Decision Tree

Definition: Given a data of attributes together with its classes, a decision tree produces a sequence of rules that can be used to classify the data.

Advantages: Decision Tree is simple to understand and visualise, requires little data preparation, and can handle both numerical and categorical data.

Disadvantages: Decision tree can create complex trees that do not generalise well, and decision trees can be unstable because small variations in the data might result in a completely different tree being generated.



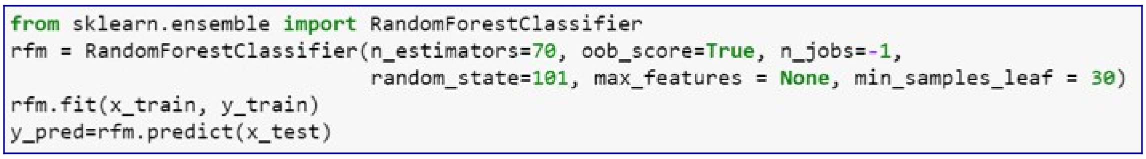
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(max_depth=10, random_state=101,
                               max_features = None, min_samples_leaf = 15)
dtree.fit(x_train, y_train)
y_pred=dtree.predict(x_test)
```

6. Random Forest

Definition: Random forest classifier is a meta-estimator that fits a number of decision trees on various sub-samples of datasets and uses average to improve the predictive accuracy of the model and controls over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement.

Advantages: Reduction in over-fitting and random forest classifier is more accurate than decision trees in most cases.

Disadvantages: Slow real time prediction, difficult to implement, and complex algorithm.



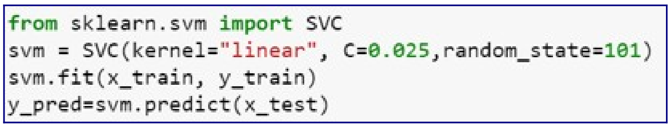
```
from sklearn.ensemble import RandomForestClassifier
rfm = RandomForestClassifier(n_estimators=70, oob_score=True, n_jobs=-1,
                             random_state=101, max_features = None, min_samples_leaf = 30)
rfm.fit(x_train, y_train)
y_pred=rfm.predict(x_test)
```

7. Support Vector Machine

Definition: Support vector machine is a representation of the training data as points in space separated into categories by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Advantages: Effective in high dimensional spaces and uses a subset of training points in the decision function so it is also memory efficient.

Disadvantages: The algorithm does not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.



```
from sklearn.svm import SVC
svm = SVC(kernel="linear", C=0.025,random_state=101)
svm.fit(x_train, y_train)
y_pred=svm.predict(x_test)
```

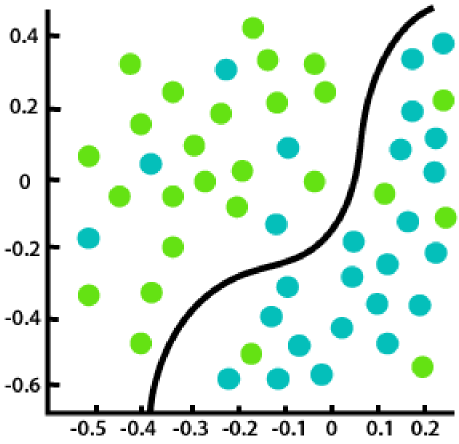
Q3. Explain the differences between regression and classification ML?

Classification:

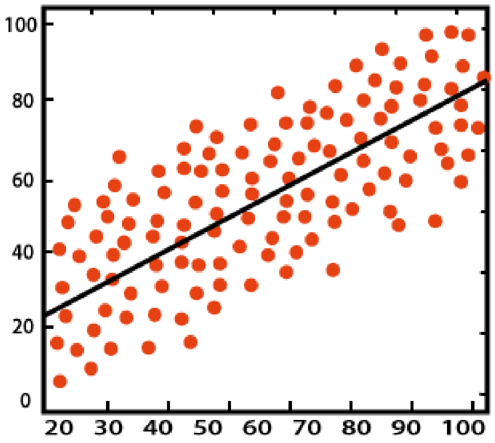
Classification is a process of finding a function which helps in dividing the dataset into classes based on different parameters. In Classification, a computer program is trained on the training dataset and based on that training, it categorizes the data into different classes.

The task of the classification algorithm is to find the mapping function to map the input(x) to the discrete output(y).

Example: The best example to understand the Classification problem is Email Spam Detection. The model is trained on the basis of millions of emails on different parameters, and whenever it receives a new email, it identifies whether the email is spam or not. If the email is spam, then it is moved to the Spam folder.



Classification



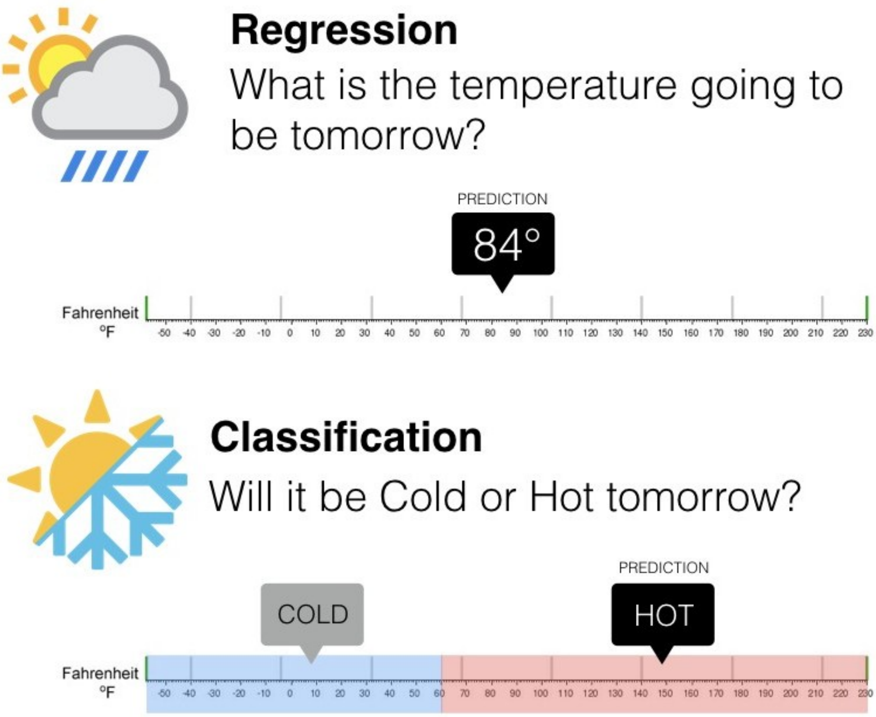
Regression

Regression:

Regression is a process of finding the correlations between dependent and independent variables. It helps in predicting the continuous variables such as prediction of **Market Trends**, prediction of House prices, etc.

The task of the Regression algorithm is to find the mapping function to map the input variable(x) to the continuous output variable(y).

Example: Suppose we want to do weather forecasting, so for this, we will use the Regression algorithm. In weather prediction, the model is trained on the past data, and once the training is completed, it can easily predict the weather for future days.



Comparison

Paramenter	**CLASSIFICATION**	**REGRESSION**
Basic	Mapping Function is used for mapping of values to predefined classes.	Mapping Function is used for mapping of values to continuous output.
Output value	Discrete values	Continuous values
Nature of the predicted data	Unordered	Ordered
Method of calculation	by measuring accuracy	by measurement of root mean square error
Example Algorithms	Decision tree, logistic regression, etc.	Regression tree (Random forest), Linear regression, etc.

Q4. Explain bias-varience trade off?

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn’t seen before. As a result, such models perform very well on training data but has high error rates on test data.

A best model will have Low Bias(low training error) and Low Variance(low testing error)

ideal situation - best model

Low Training Error and Low Evaluation Error

Other situations

Low Training Error and High Evaluation Error

- This situation is known as Overfitting.
- The model is well trained. but not explained every possibilities
- The model is not generic

High Training Error and High Evaluation Error

- under-fitting
- Change the algorithm (this is the best option)
- Use more data field/data

High training error and low evaluation error - void situation

Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalise.

Right Model - Low Bias,Low Variance
Under-fit - High Bias,Low Variance
Overfit - Low Bias, High Variance

How to handle under-fitting

Use better algorithms
Use more data

How to handle Overfitting

(we have enough data and the model is also good)
There are many techniques to avoid/minimise overfitting

1. Regularisation - mostly used for linear models
    a.Ridge Regression
    b.Lasso Regression
    c.Elastic Net
2.Ensemble Models - used for non-linear models
    a.Bagging - Random Forest algorithm
    b.Boosting - XG Boost algorithm,Ada boost algorithm

Q5. Explain confusion matrix? Also explain terms True positive, True negative, False positive and false negative?

A confusion matrix is a table 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.

True Positive (TP)

- The predicted value matches the actual value
- The actual value was positive and the model predicted a positive value

True Negative (TN)

- The predicted value matches the actual value

- The actual value was negative and the model predicted a negative value

False Positive (FP) – Type 1 error

- The predicted value was falsely predicted
- The actual value was negative but the model predicted a positive value
- Also known as the **Type 1 error**

False Negative (FN) – Type 2 error

- The predicted value was falsely predicted
- The actual value was positive but the model predicted a negative value
- Also known as the **Type 2 error**

Q6. What is ROC curve and what does it represent?

An **ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

True Positive Rate (**TPR**) is a synonym for recall and is therefore defined as follows:

$$TPR = \frac{TP}{TP+FN}$$

False Positive Rate (**FPR**) is defined as follows:

$$FPR = \frac{FP}{FP+TN}$$

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

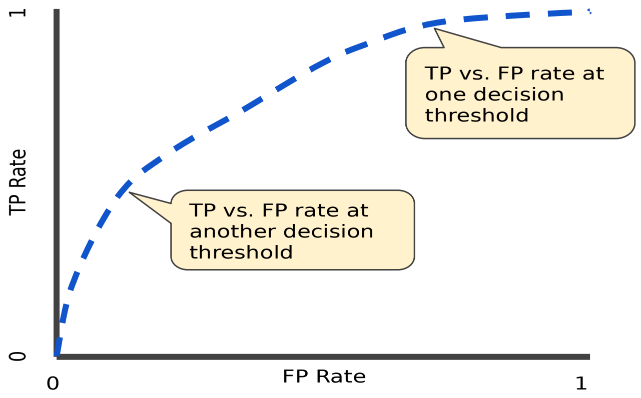
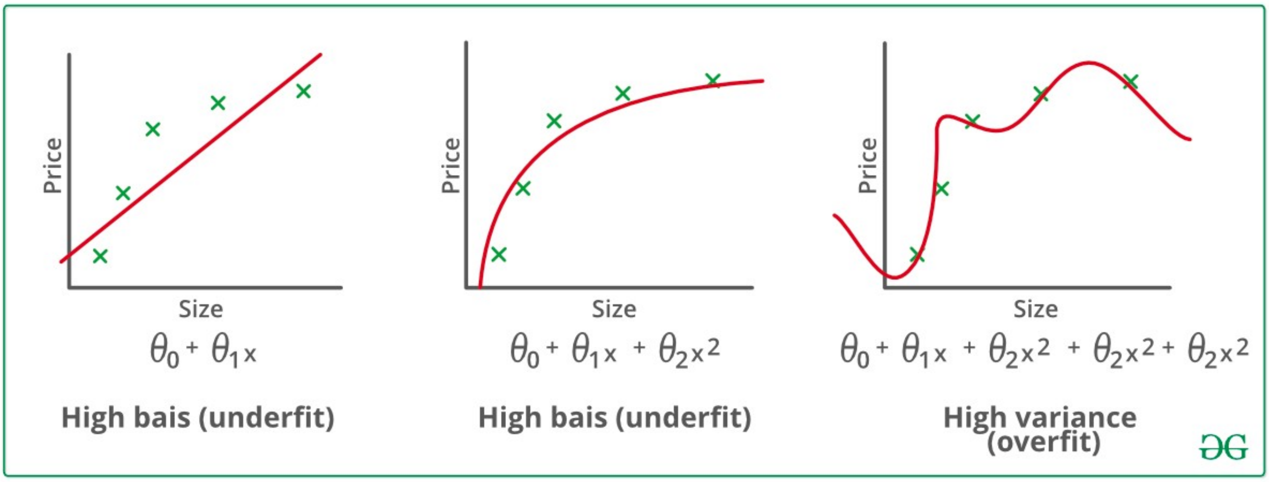


Fig: TP vs. FP rate at different classification thresholds.

Q7. What are the differences between over-fitting and under-fitting?

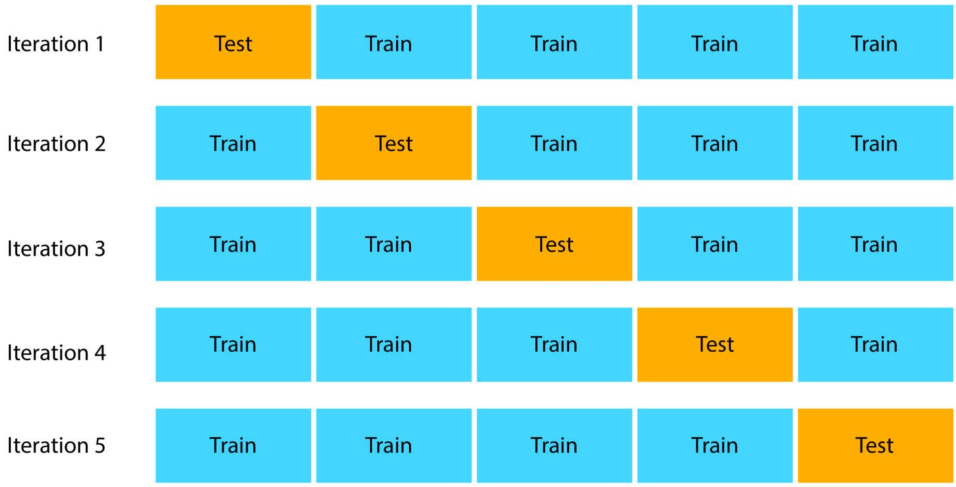
Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model.

Underfitting means, incomplete information or the minimum number of features are given. Like if the only shape is given then model faces the problem to identify that it is a ball or any round shape fruit.



Q8. Explain cross validation concept?

To tackle Problem of Overfitting, the answer is Cross Validation. Cross validation is a statistical method used to estimate the performance (or accuracy) of machine learning models. When dealing with a Machine Learning task, we have to properly identify the problem so that you can pick the most suitable algorithm which can give you the best score.



Iteration 1	Test	Train	Train	Train	Train
Iteration 2	Train	Test	Train	Train	Train
Iteration 3	Train	Train	Test	Train	Train
Iteration 4	Train	Train	Train	Test	Train
Iteration 5	Train	Train	Train	Train	Test

K fold cross validation

K-fold cross validation is one way to improve the holdout method. This method guarantees that the score of our model does not depend on the way we picked the train and test set. The data set is divided into k number of subsets and the holdout method is repeated k number of times. Let us go through this in steps:

1. Randomly split your entire dataset into k number of folds (subsets)
2. For each fold in your dataset, build your model on k – 1 folds of the dataset. Then, test the model to check the effectiveness for kth fold
3. Repeat this until each of the k-folds has served as the test set
4. The average of your k recorded accuracy is called the cross-validation accuracy and will serve as your performance metric for the model.

The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation.

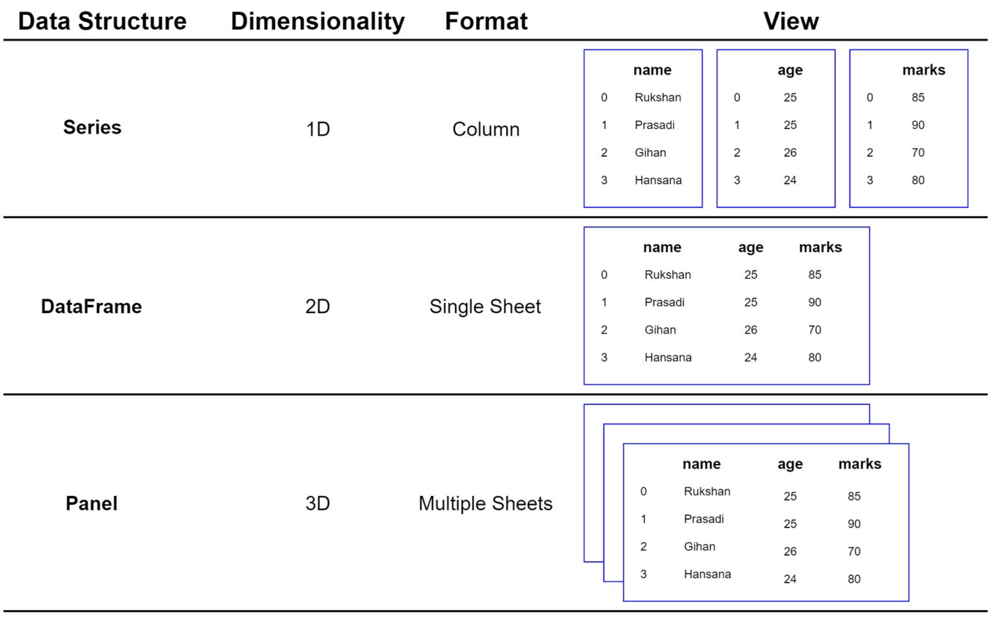
Assume that the data will be split into k chunks, say 10 chunks.
The model will be trained in 10 iterations using 9 chunks for training and remaining 1 for testing.

S1- training 1-9 evaluation 10
S2 - training 2-10 evaluation 1
S3 - training 3-10,1 evaluation 2
S10 - training 1-8,10 evaluation 9

Q9. What Are the Different Types Of Data Structures In Pandas?

A **data structure** is a collection of data values and defines the relationship between the data, and the operations that can be performed on the data. There are three main data structures in pandas:

- **Series** — 1D
- **DataFrame** — 2D
- **Panel** — 3D



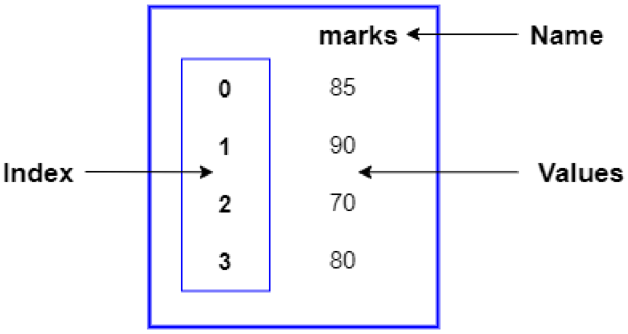
Data Structure	Dimensionality	Format	View
Series	1D	Column	name 0 Rukshan 1 Prasadi 2 Gihan 3 Hansana age 0 25 1 25 2 26 3 24 marks 0 85 1 90 2 70 3 80
DataFrame	2D	Single Sheet	name age marks 0 Rukshan 25 85 1 Prasadi 25 90 2 Gihan 26 70 3 Hansana 24 80
Panel	3D	Multiple Sheets	name age marks 0 Rukshan 25 85 1 Prasadi 25 90 2 Gihan 26 70 3 Hansana 24 80

Image Copyright: Rukshan Pramoditha

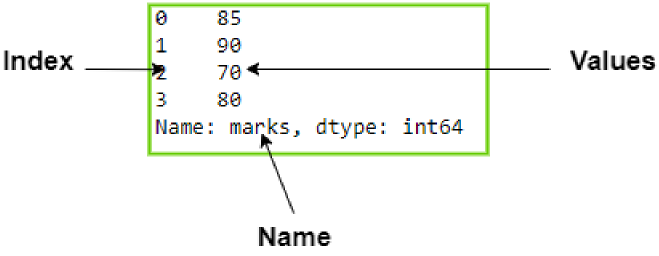
The pandas Series

The ***Series*** is the object of the pandas library designed to represent one-dimensional data structures, similar to an array but with some additional features. A series consists of two components.

- **One-dimensional data (Values)**
- **Index**



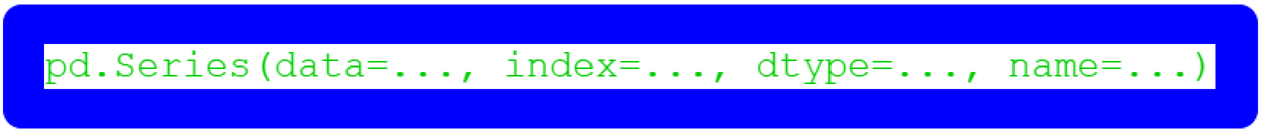
pd.Series([85, 90, 70, 80], name='marks')



```
0     85
1     90
2     70
3     80
Name: marks, dtype: int64
```

Image Copyright: Rukshan Pramoditha

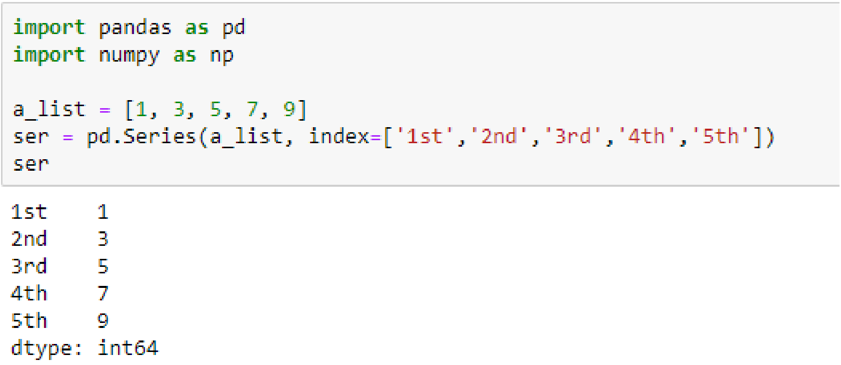
The general construct for creating a Series data structure is:



```
pd.Series(data=..., index=..., dtype=..., name=...)
```

Series creation: Using a Python list

To create a series using a Python list, you can just pass a list to the ***data*** parameter of the **Series()** class constructor.

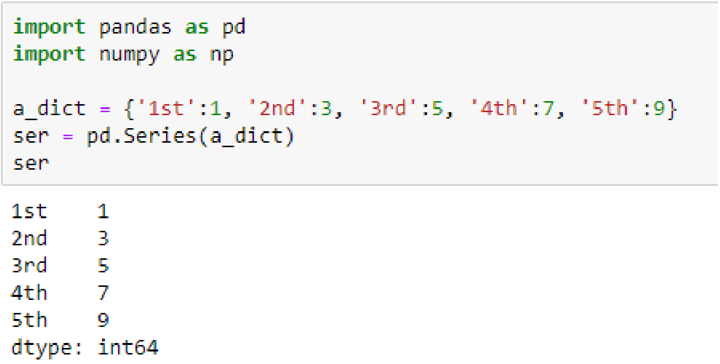


```
import pandas as pd
import numpy as np

a_list = [1, 3, 5, 7, 9]
ser = pd.Series(a_list, index=['1st','2nd','3rd','4th','5th'])
ser
```

```
1st    1
2nd    3
3rd    5
4th    7
5th    9
dtype: int64
```

Series creation: Using a Python dictionary

To create a series using a Python dictionary, you can just pass a dictionary to the ***data*** parameter of the **Series()** class constructor. This time, the arrays of the index and values are filled with the corresponding keys and values of the dictionary.

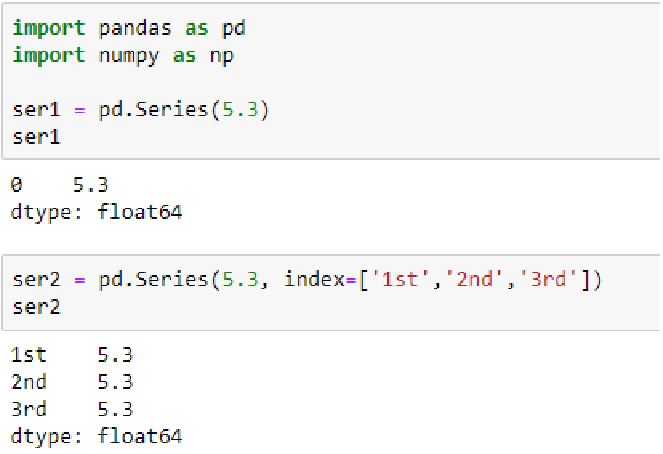


```
import pandas as pd
import numpy as np

a_dict = {'1st':1, '2nd':3, '3rd':5, '4th':7, '5th':9}
ser = pd.Series(a_dict)
ser
```

```
1st    1
2nd    3
3rd    5
4th    7
5th    9
dtype: int64
```

Series creation: Using a scalar value

we can also create a Series from a scalar value. If you do not specify the ***index*** argument, the default index is 0. If you specify the ***index***, the value will be repeated for specified index values.



```
import pandas as pd
import numpy as np

ser1 = pd.Series(5.3)
ser1
```

```
0    5.3
dtype: float64
```

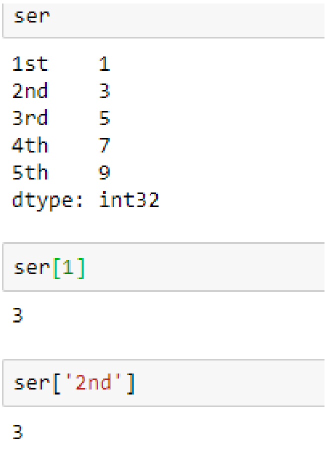
```
ser2 = pd.Series(5.3, index=['1st','2nd','3rd'])
ser2
```

```
1st    5.3
2nd    5.3
3rd    5.3
dtype: float64
```

Selecting elements from a Series

The indexing and slicing that are applicable to NumPy arrays can be extended to the series because pandas library was built on top of NumPy.

- You can select a single element using the index number or index label of the series.



```
ser
```

```
1st    1
2nd    3
3rd    5
4th    7
5th    9
dtype: int32
```

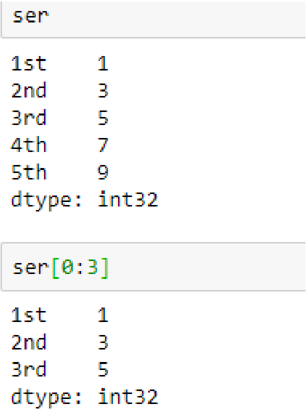
```
ser[1]
```

```
3
```

```
ser['2nd']
```

```
3
```

- You can use the slice **:** notation with index numbers to select a range of elements from a series.



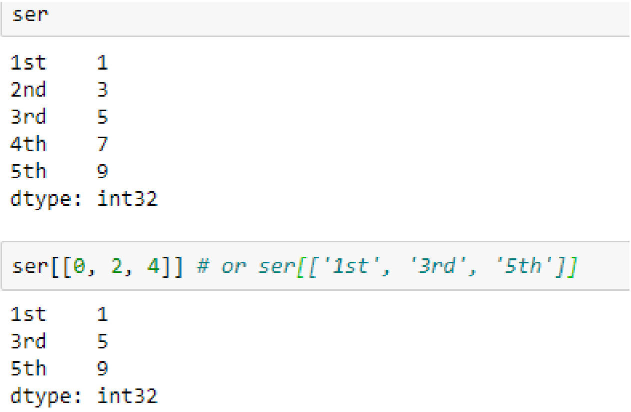
```
ser
```

```
1st    1
2nd    3
3rd    5
4th    7
5th    9
dtype: int32
```

```
ser[0:3]
```

```
1st    1
2nd    3
3rd    5
dtype: int32
```

- You can also use a list of index numbers or index labels to select multiple elements from a series.



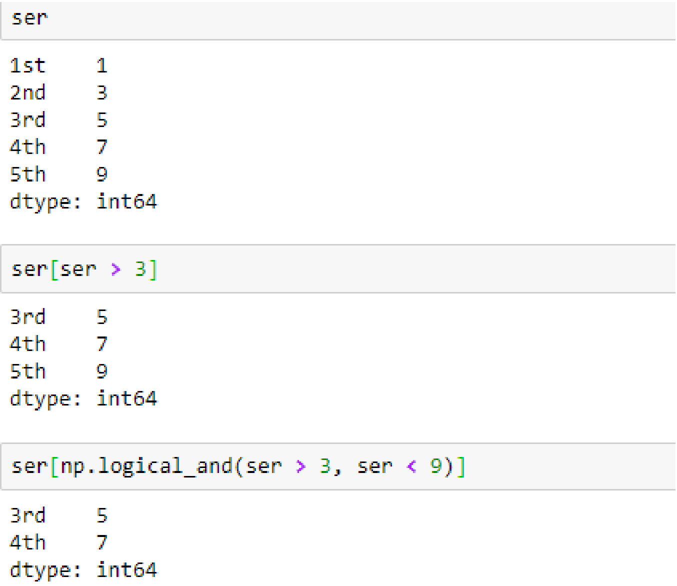
```
ser
```

```
1st    1
2nd    3
3rd    5
4th    7
5th    9
dtype: int32
```

```
ser[[0, 2, 4]] # or ser[['1st', '3rd', '5th']]
```

```
1st    1
3rd    5
5th    9
dtype: int32
```

- You can also use the conditions and Boolean operators to select elements from a series.



```
ser
```

```
1st    1
2nd    3
3rd    5
4th    7
5th    9
dtype: int64
```

```
ser[ser > 3]
```

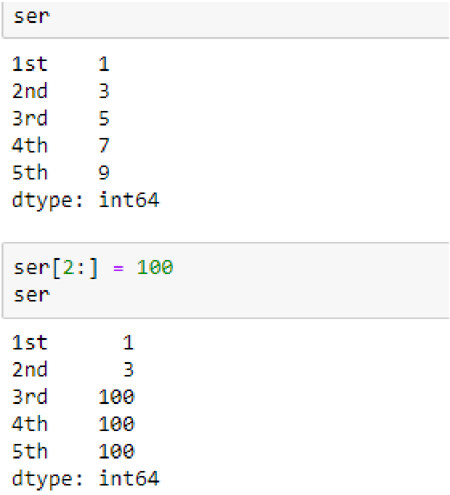
```
3rd    5
4th    7
5th    9
dtype: int64
```

```
ser[np.logical_and(ser > 3, ser < 9)]
```

```
3rd    5
4th    7
dtype: int64
```

Assigning values to the elements

Series are **mutable**, which means that you can change the value of an element in the series after it has been initialized.



```
ser
```

```
1st    1
2nd    3
3rd    5
4th    7
5th    9
dtype: int64
```

```
ser[2:] = 100
ser
```

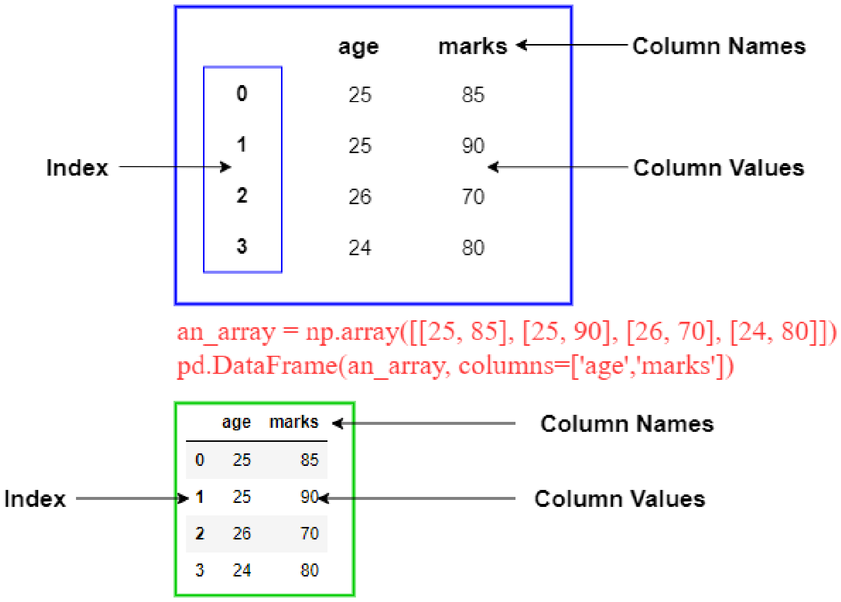
```
1st      1
2nd      3
3rd    100
4th    100
5th    100
dtype: int64
```

The pandas DataFrame

A ***DataFrame*** is a two-dimensional data structure composed of rows and columns. *Each column of a DataFrame is a pandas Series*. These columns should be of the same length, but they can be of different data types — float, int, bool, and so on. DataFrames are both ***value-mutable*** and ***size-mutable*** (***Series***, by contrast, is only value-mutable, not size-mutable. The length of a Series cannot be changed although the values can be changed). This lets us perform operations that would alter values held within the DataFrame or add/delete columns to/from the DataFrame.

A DataFrame consists of three components.

- **Two-dimensional data (Values)**
- **Row index**
- **Column index**



	age	marks
0	25	85
1	25	90
2	26	70
3	24	80

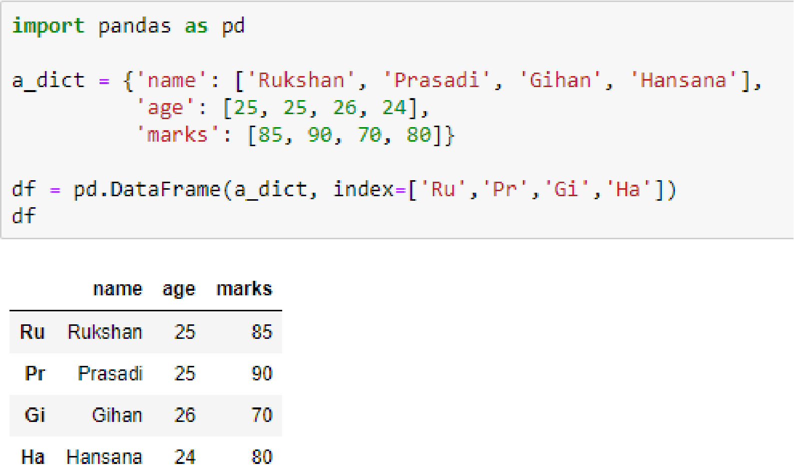
```
an_array = np.array([[25, 85], [25, 90], [26, 70], [24, 80]])
pd.DataFrame(an_array, columns=['age','marks'])
```

Image Copyright: Rukshan Pramoditha

The general construct for creating a DataFrame data structure is:

```
pd.DataFrame(data=..., index=..., columns=...)
```

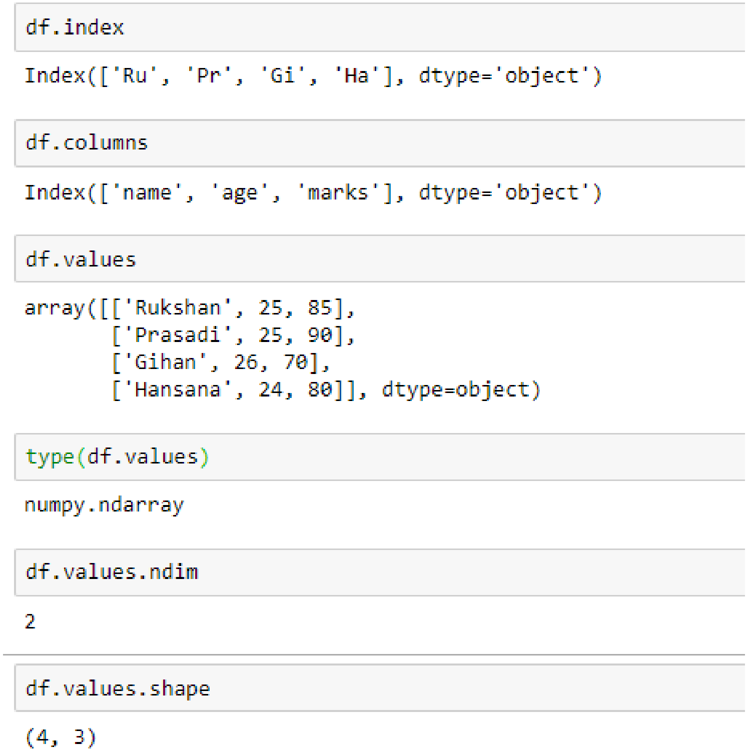
DataFrame creation: Using a dictionary lists



```
import pandas as pd

a_dict = {'name': ['Rukshan', 'Prasadi', 'Gihan', 'Hansana'],
          'age': [25, 25, 26, 24],
          'marks': [85, 90, 70, 80]}

df = pd.DataFrame(a_dict, index=['Ru','Pr','Gi','Ha'])
df
```

	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

If you want to see the individual components which make up the DataFrame, you can call ***values***, ***index*** and ***columns*** attributes of the DataFrame.



```
df.index
```

```
Index(['Ru', 'Pr', 'Gi', 'Ha'], dtype='object')
```

```
df.columns
```

```
Index(['name', 'age', 'marks'], dtype='object')
```

```
df.values
```

```
array([['Rukshan', 25, 85],
       ['Prasadi', 25, 90],
       ['Gihan', 26, 70],
       ['Hansana', 24, 80]], dtype=object)
```

```
type(df.values)
```

```
numpy.ndarray
```

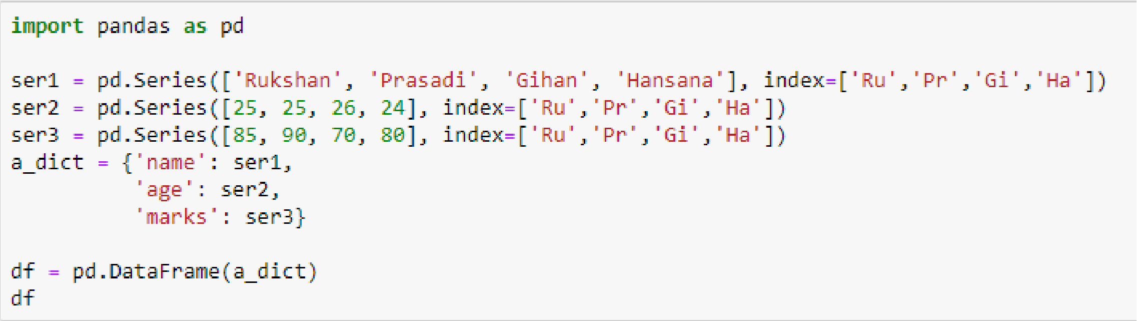
```
df.values.ndim
```

```
2
```

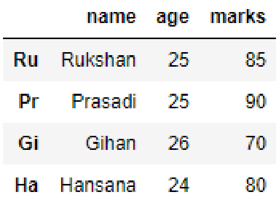
```
df.values.shape
```

```
(4, 3)
```

DataFrame creation: Using a dictionary of series



```
import pandas as pd

ser1 = pd.Series(['Rukshan', 'Prasadi', 'Gihan', 'Hansana'], index=['Ru','Pr','Gi','Ha'])
ser2 = pd.Series([25, 25, 26, 24], index=['Ru','Pr','Gi','Ha'])
ser3 = pd.Series([85, 90, 70, 80], index=['Ru','Pr','Gi','Ha'])
a_dict = {'name': ser1,
          'age': ser2,
          'marks': ser3}

df = pd.DataFrame(a_dict)
df
```



	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

DataFrame creation: Using pandas read_*

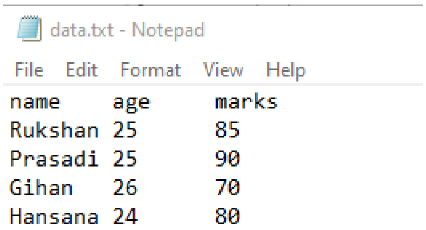
pandas supports many different file formats or data sources out of the box (csv, excel, sql, json, …), each of them with the prefix read_*

pandas **read_csv()** function: Reads a comma-separated values (csv) file or a text file into a pandas DataFrame.

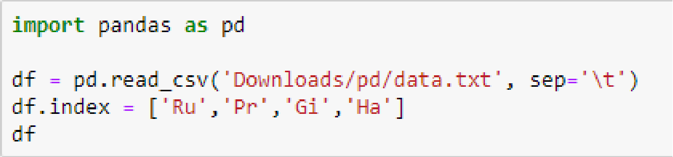
- pandas **read_excel()** function: Reads an Excel file into a pandas DataFrame.
- pandas **read_html()** function: Reads HTML tables.
- pandas **read_sql()** function: Read SQL query or database table into a DataFrame.

The syntax of using **read_csv()** is shown in the following code.

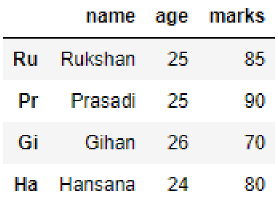
```
pd.read_csv(filepath, sep=', ', dtype=None, header=None, names=None,
skiprows=None, index_col=None, skip_blank_lines=TRUE, na_filter=TRUE)
```



```
name    age     marks
Rukshan 25      85
Prasadi 25      90
Gihan   26      70
Hansana 24      80
```



```
import pandas as pd

df = pd.read_csv('Downloads/pd/data.txt', sep='\t')
df.index = ['Ru','Pr','Gi','Ha']
df
```

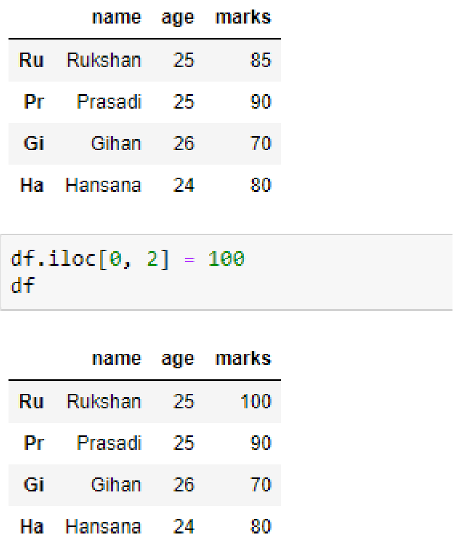


	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

Assigning values to the elements and adding new columns

DataFrames are both ***value-mutable*** and ***size-mutable***. This means that you can change values held within the DataFrame or add/delete columns to/from the DataFrame.

Value mutability (changing values)

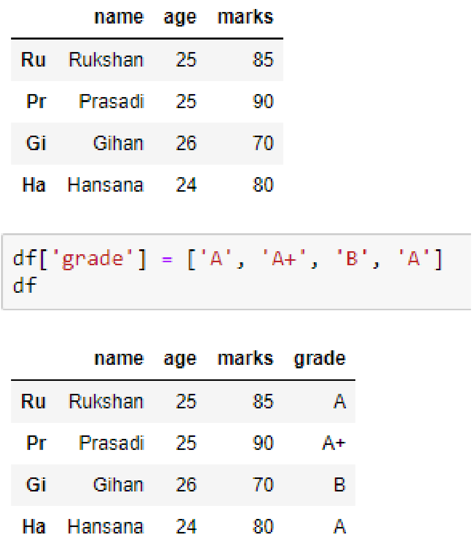


	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

```
df.iloc[0, 2] = 100
df
```

	name	age	marks
Ru	Rukshan	25	100
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

Size mutability (Adding new rows and columns)

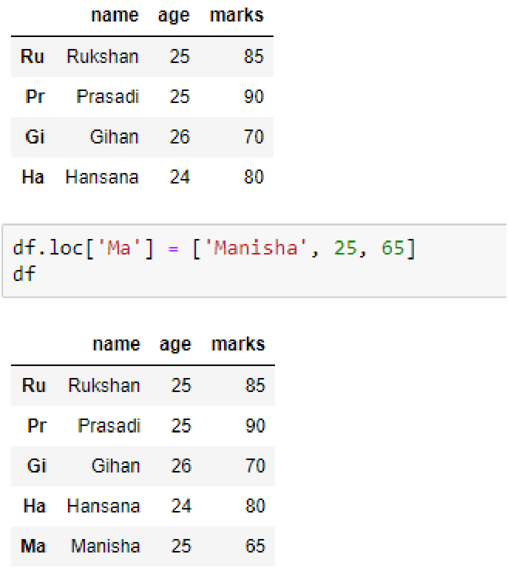


	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

```
df['grade'] = ['A', 'A+', 'B', 'A']
df
```

	name	age	marks	grade
Ru	Rukshan	25	85	A
Pr	Prasadi	25	90	A+
Gi	Gihan	26	70	B
Ha	Hansana	24	80	A

Adding a new column



	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

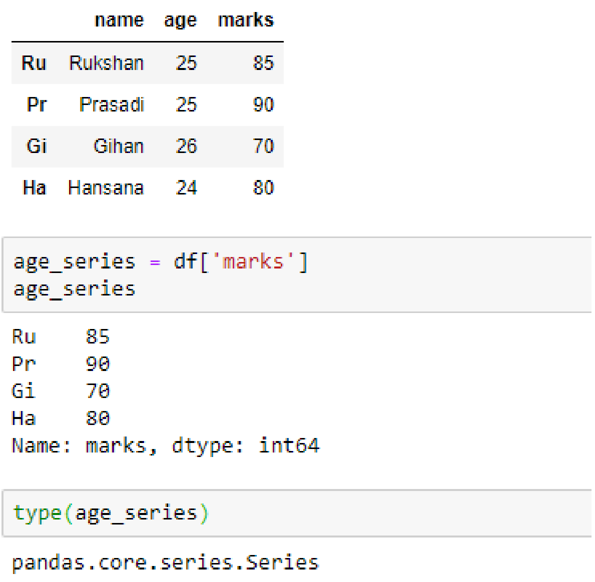
```
df.loc['Ma'] = ['Manisha', 25, 65]
df
```

	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80
Ma	Manisha	25	65

Adding new row

Each column in a `DataFrame` is a `Series`

If you're just interested in working with the data in the column **marks**, you can extract it as a series.



	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

```
age_series = df['marks']
age_series
```

```
Ru    85
Pr    90
Gi    70
Ha    80
Name: marks, dtype: int64
```

```
type(age_series)
```

```
pandas.core.series.Series
```

The pandas Panel

A Panel is a 3D array. It is not as widely used as Series or DataFrames. It is not as easily displayed on screen or visualized as the other two because of its 3D nature. It is generally used for 3D time-series data. The three-axis names are as follows:

- **items:** This is axis 0. Each item corresponds to a DataFrame structure.
- **major_axis:** This is axis 1. Each item corresponds to the rows of the DataFrame structure.
- **minor_axis:** This is axis 2. Each item corresponds to the columns of each DataFrame structure.

Q10. Explain Series and DataFrame In Pandas.

The *Series* is the object of the pandas library designed to represent one-dimensional data structures, similar to an array but with some additional features. A series consists of two components.

- **One-dimensional data (Values)**
- **Index**

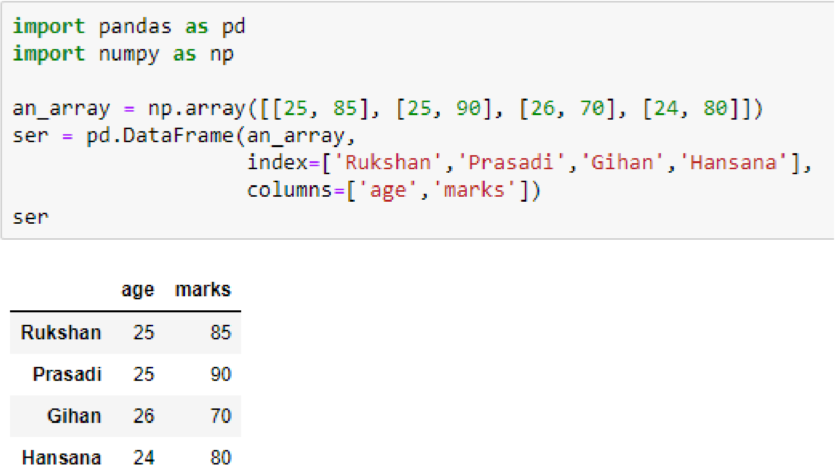
A *DataFrame* is a two-dimensional data structure composed of rows and columns — exactly like a simple spreadsheet or a SQL table. *Each column of a DataFrame is a pandas Series*. These columns should be of the same length, but they can be of different data types — float, int, bool, and so on.

A DataFrame consists of three components.

- **Two-dimensional data (Values)**
- **Row index**
- **Column index**

Q11. What are the different ways of creating DataFrame in pandas? Explain with examples.

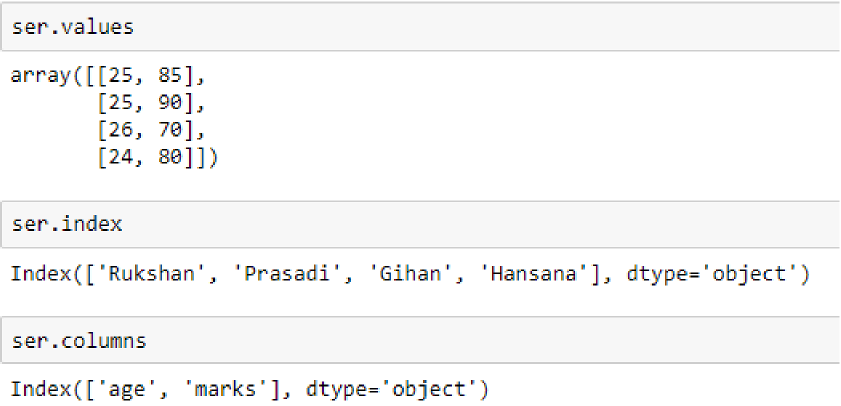
DataFrame creation: Using a two-dimensional ndarray



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

an_array = np.array([[25, 85], [25, 90], [26, 70], [24, 80]])
ser = pd.DataFrame(an_array,
                   index=['Rukshan','Prasadi','Gihan','Hansana'],
                   columns=['age','marks'])
ser
```

	age	marks
Rukshan	25	85
Prasadi	25	90
Gihan	26	70
Hansana	24	80

If you want to see the individual components which make up the DataFrame, you can call ***values***, ***index*** and ***columns*** attributes of the DataFrame.



```python
ser.values
```

```
array([[25, 85],
       [25, 90],
       [26, 70],
       [24, 80]])
```

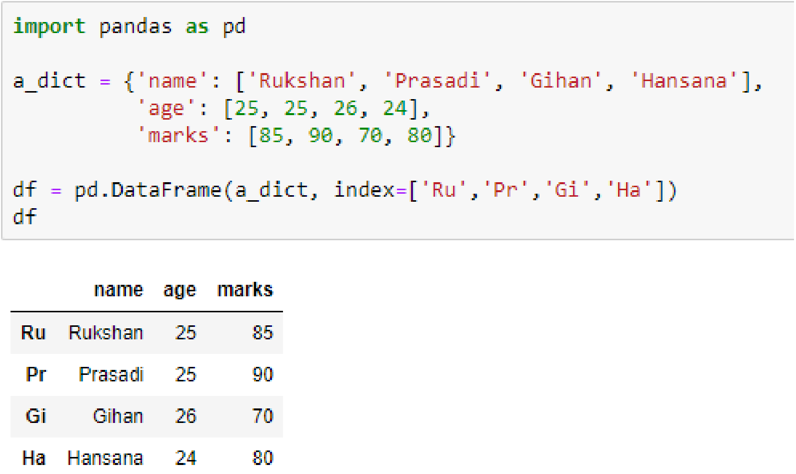
```python
ser.index
```

```
Index(['Rukshan', 'Prasadi', 'Gihan', 'Hansana'], dtype='object')
```

```python
ser.columns
```

```
Index(['age', 'marks'], dtype='object')
```

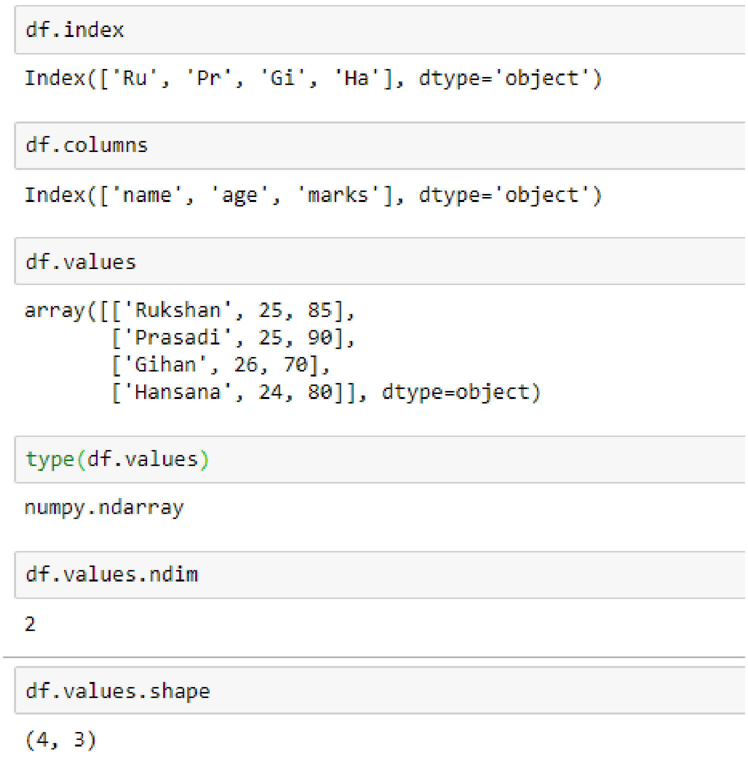
DataFrame creation: Using a dictionary lists



```python
import pandas as pd

a_dict = {'name': ['Rukshan', 'Prasadi', 'Gihan', 'Hansana'],
          'age': [25, 25, 26, 24],
          'marks': [85, 90, 70, 80]}

df = pd.DataFrame(a_dict, index=['Ru','Pr','Gi','Ha'])
df
```

	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

If you want to see the individual components which make up the DataFrame, you can call *values*, *index* and *columns* attributes of the DataFrame.



```
df.index
```

```
Index(['Ru', 'Pr', 'Gi', 'Ha'], dtype='object')
```

```
df.columns
```

```
Index(['name', 'age', 'marks'], dtype='object')
```

```
df.values
```

```
array([['Rukshan', 25, 85],
       ['Prasadi', 25, 90],
       ['Gihan', 26, 70],
       ['Hansana', 24, 80]], dtype=object)
```

```
type(df.values)
```

```
numpy.ndarray
```

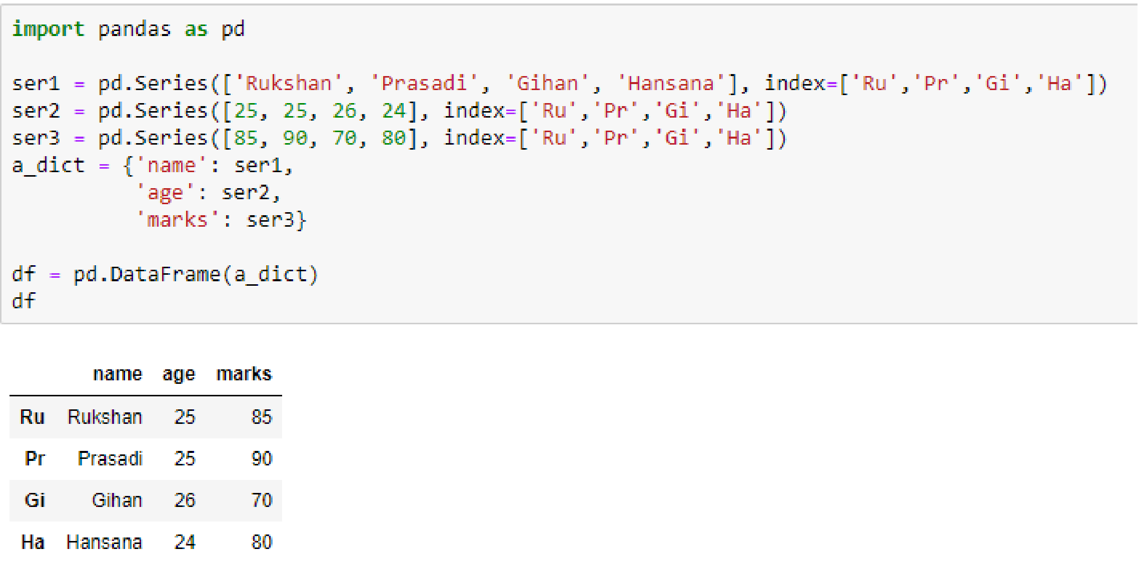
```
df.values.ndim
```

```
2
```

```
df.values.shape
```

```
(4, 3)
```

DataFrame creation: Using a dictionary of series



```
import pandas as pd

ser1 = pd.Series(['Rukshan', 'Prasadi', 'Gihan', 'Hansana'], index=['Ru','Pr','Gi','Ha'])
ser2 = pd.Series([25, 25, 26, 24], index=['Ru','Pr','Gi','Ha'])
ser3 = pd.Series([85, 90, 70, 80], index=['Ru','Pr','Gi','Ha'])
a_dict = {'name': ser1,
          'age': ser2,
          'marks': ser3}

df = pd.DataFrame(a_dict)
df
```

	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

DataFrame creation: Using pandas read_*

pandas supports many different file formats or data sources out of the box (csv, excel, sql, json, …), each of them with the prefix read_*

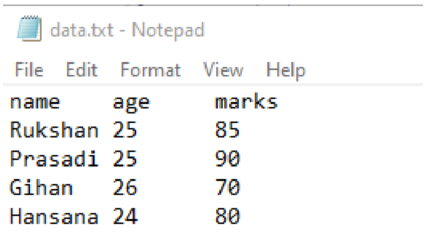
pandas **read_csv()** function: Reads a comma-separated values (csv) file or a text file into a pandas DataFrame.

- pandas **read_excel()** function: Reads an Excel file into a pandas DataFrame.

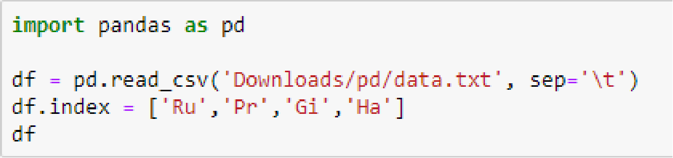
- pandas **read_html()** function: Reads HTML tables.
- pandas **read_sql()** function: Read SQL query or database table into a DataFrame.

The syntax of using **read_csv()** is shown in the following code.

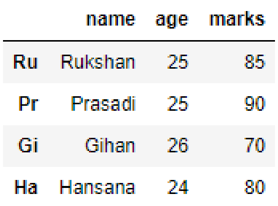
```
pd.read_csv(filepath, sep=', ', dtype=None, header=None, names=None,
skiprows=None, index_col=None, skip_blank_lines=TRUE, na_filter=TRUE)
```



```
name	age	marks
Rukshan	25	85
Prasadi	25	90
Gihan	26	70
Hansana	24	80
```



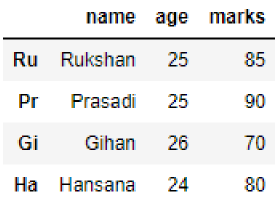
```
import pandas as pd

df = pd.read_csv('Downloads/pd/data.txt', sep='\t')
df.index = ['Ru','Pr','Gi','Ha']
df
```



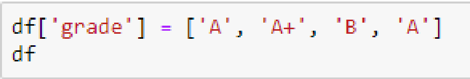
	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

Q12. How Will You Add An Index, Row, Or Column To A Dataframe In Pandas?

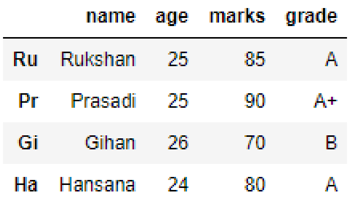
Size mutability (Adding new rows and columns)



	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80

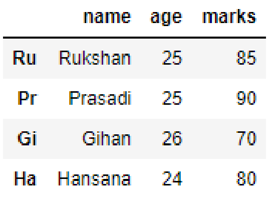


```
df['grade'] = ['A', 'A+', 'B', 'A']
df
```

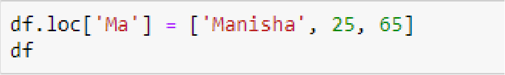


	name	age	marks	grade
Ru	Rukshan	25	85	A
Pr	Prasadi	25	90	A+
Gi	Gihan	26	70	B
Ha	Hansana	24	80	A

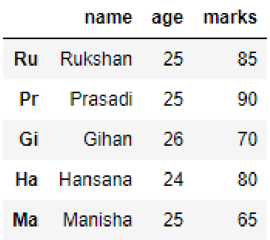
Adding a new column



	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80



```
df.loc['Ma'] = ['Manisha', 25, 65]
df
```



	name	age	marks
Ru	Rukshan	25	85
Pr	Prasadi	25	90
Gi	Gihan	26	70
Ha	Hansana	24	80
Ma	Manisha	25	65

Adding new row

Q13. How Can You Iterate Over Dataframe In Pandas?

We can access data of DataFrames in many ways,

1. Value
2. Columns
3. Rows

We are creating a DataFrame using the following commands. Then, we will see how we can access its rows, columns, & values and understand how it can be done in different ways.

```
##Creating DataFrame
import pandas as pd
dictObj = {'EmpId' : ['E01','E02','E03','E04'], 'EmpName' : ['Raj','Atul','Reena','Ayushi'],
'Department' : ['IT','IT','HR','Accounts']}
df=pd.DataFrame(dictObj, index=['First','Second','Third','Fourth'])
```

##Output

```
              EmpId    EmpName   Department

     First     E01        Raj        IT
     Second    E02        Atul       IT
     Third     E03        Reena      HR
     Fourth    E04        Ayushi     Accounts
```

1. Value

We can access the individual value of DataFrame in the following ways.

By using row name and row index number

Using the row name and row index number along with the column, we can easily access a single value of a DataFrame.

Syntax : <DataFrame Object> . <column name> [<row name/ row index number>]

Example :

```
df.EmpName['Third']   ##Access using row name
##Output  Reena

df.EmpName[2]       ## Access using row index
##Output Reena
```

By using at and iat attributes

We can also access a single value of a DataFrame with the help of “at” and “iat” attributes.

Syntax	Purpose
<DataFrame Object>.at [<row name>,<column name>]	Access a single value by row/column name.
<DataFrame Object>.iat [<row index no>,<column index no>]	Access a single value by row/column index no

Example

```
df.at[‘Second’,’EmpName’]
##Output   Atul

df.iat[2,2]
## Output   HR
```

Accessing Single Column

We can also access column of a DataFrame by using the below syntax,

```
<DataFrame Object> [ <Column Name> ]
<DataFrame Object> . <Column Name>
```

Like

```
df [‘EmpName’]    ## First Way

df.EmpName    ## Second Way
```

Accessing Multiple Columns

We can also access multiple columns of a DataFrame by passing a list of columns name inside the square bracket.

Syntax : <DataFrame Object> [[<Column Name1> , <Column Name2>,<Column Name3>]]

Example : df [[‘EmpName’,’Department’]]

Accessing Rows

We can access rows of a DataFrame in two ways.

- By using loc and iloc

We can access a single row and multiple rows of a DataFrame with the help of “loc” and “iloc”.

Syntax	Purpose
<DataFrame Object>.loc [[<row name>]]	Access a single row or multiple rows by name.
<DataFrame Object>.iloc [[<row index no>]]	Access a single or multiple rows by row index no

```
df.loc[[‘Second’]]   ## Access row using location, pass row name
df.iloc[[2]]          ## Access row using row index number
```

Adding a Row

at and loc

```
##Syntax :
<DataFrame Object> . at [<row index name> ] = <values>
<DataFrame Object> . loc [<row index name> ] = <values>

##Example:
df.at[‘Fifth’, : ] = [‘E05’,’Nakul’,’HR’]
df.at[‘Sixth’]=[‘E06’,’Rahul’,’Accounts’]
df.at[‘Seventh’,’EmpName’:’Department’] = [‘Vipul’,’IT’]
```

Modifying a Row

To change the value of a row, we can use the below syntax.

```
<DataFrame Object>.<Column Name> [<row name>] = <new value>  ##Syntax

##Example
df.EmpId[‘Seventh’]=’E07’
```

Adding a column

It is very easy to add a column into an existing DataFrame. We can use the below syntax for adding a column into DataFrame.

```
<DataFrame Object> [<Column Name>]  ##Syntax

##Examples
df[‘City’]  ## added a new column with ‘NaN’ as default value.
df[‘City’]=’New Delhi’
```

Deleting a Column

Del is used to delete a column from DataFrame.

```
del <DataFrame Object> [<Column name>]   ##Syntax

del df ['City']  ##Example
```

Q14. How Can A Dataframe Be Converted To An Excel File?

Algorithm:

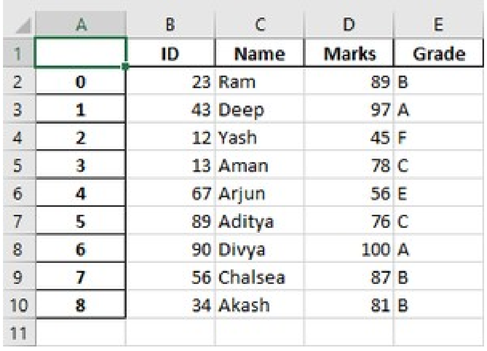
1. Create the DataFrame.
2. Determine the name of the Excel file.
3. Call **to_excel()** function with the file name to export the DataFrame.

Example 1:

```
# importing the module
import pandas as pd
```

```
# creating the DataFrame
marks_data = pd.DataFrame({'ID': {0: 23, 1: 43, 2: 12,
                                  3: 13, 4: 67, 5: 89,
                                  6: 90, 7: 56, 8: 34},
                           'Name': {0: 'Ram', 1: 'Deep',
                                  2: 'Yash', 3: 'Aman',
                                  4: 'Arjun', 5: 'Aditya',
                                  6: 'Divya', 7: 'Chalsea',
                                  8: 'Akash' },
                           'Marks': {0: 89, 1: 97, 2: 45, 3: 78,
                                  4: 56, 5: 76, 6: 100, 7: 87,
                                  8: 81},
                           'Grade': {0: 'B', 1: 'A', 2: 'F', 3: 'C',
                                  4: 'E', 5: 'C', 6: 'A', 7: 'B',
                                  8: 'B'}})

# determining the name of the file
file_name = 'MarksData.xlsx'

# saving the excel
marks_data.to_excel(file_name)
print('DataFrame is written to Excel File successfully.')
```

The Excel file is:



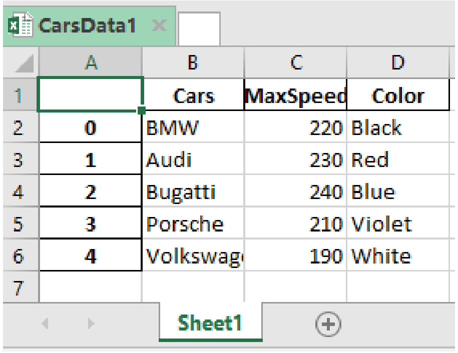
	A	B	C	D	E
1		ID	Name	Marks	Grade
2	0	23	Ram	89	B
3	1	43	Deep	97	A
4	2	12	Yash	45	F
5	3	13	Aman	78	C
6	4	67	Arjun	56	E
7	5	89	Aditya	76	C
8	6	90	Divya	100	A
9	7	56	Chalsea	87	B
10	8	34	Akash	81	B
11					

Example 2: We can also first use the ExcelWriter() method to save it.

```
# importing the module
import pandas as pd

# creating the DataFrame
cars_data = pd.DataFrame({'Cars': ['BMW', 'Audi', 'Bugatti',
                                   'Porsche', 'Volkswagen'],
                          ‘MaxSpeed': [220, 230, 240, 210, 190],
                          'Color': ['Black', 'Red', 'Blue',
                                    'Violet', 'White']})

# writing to Excel
datatoexcel = pd.ExcelWriter('CarsData1.xlsx')

# write DataFrame to excel
```

```
cars_data.to_excel(datatoexcel)

# save the excel
datatoexcel.save()
print('DataFrame is written to Excel File successfully.')
```

Output:



	A	B	C	D
1		Cars	MaxSpeed	Color
2	0	BMW	220	Black
3	1	Audi	230	Red
4	2	Bugatti	240	Blue
5	3	Porsche	210	Violet
6	4	Volkswag	190	White
7				

Q15. What Is Groupby Function In Pandas?

Pandas DataFrame groupby() function is used to group rows that have the same values. It’s mostly used with aggregate functions (count, sum, min, max, mean) to get the statistics based on one or more column values.

Pandas gropuby() function is very similar to the SQL group by statement. Afterall, DataFrame and SQL Table are almost similar too. It’s an intermediary function to create groups before reaching the final result.

Example:

Consider the CSV file

```
ID,Name,Role,Salary
1,Pankaj,Editor,10000
2,Lisa,Editor,8000
3,David,Author,6000
4,Ram,Author,4000
5,Anupam,Author,5000
```

We will use pandas read_csv() function to read the CSV file and create the DataFrame object.

```
import pandas as pd
df = pd.read_csv('records.csv')
print(df)
```

Average Salary Group By Role

```
df_groupby_role = df.groupby(['Role'])

# select only required columns

df_groupby_role = df_groupby_role[["Role", "Salary"]]
```

```
# get the average

df_groupby_role_mean = df_groupby_role.mean()

print(df_groupby_role_mean)
```

Output:

```
        Salary
Role
Author    5000
Editor    9000
```

16. What is Standardisation and Normalisation ?

“Normalizing” a vector most often means dividing by a norm of the vector. It also often refers to rescaling by the minimum and range of the vector, to make all the elements lie between 0 and 1 thus bringing all the values of numeric columns in the dataset to a common scale.

“Standardizing” a vector most often means subtracting a measure of location and dividing by a measure of scale. For example, if the vector contains random values with a Gaussian distribution, you might subtract the mean and divide by the standard deviation, thereby obtaining a “standard normal” random variable with mean 0 and standard deviation 1.

18. Explain the steps to remove outliers from a data?

Box Plot is the visual representation of the depicting groups of numerical data through their quartiles. Boxplot is also used for detect the outlier in data set. It captures the summary of the data efficiently with a simple box and whiskers and allows us to compare easily across groups. Boxplot summarizes a sample data using 25th, 50th and 75th percentiles. These percentiles are also known as the lower quartile, median and upper quartile.

A box plot consist of 5 things.

- Minimum
- First Quartile or 25%
- Median (Second Quartile) or 50%
- Third Quartile or 75%
- Maximum

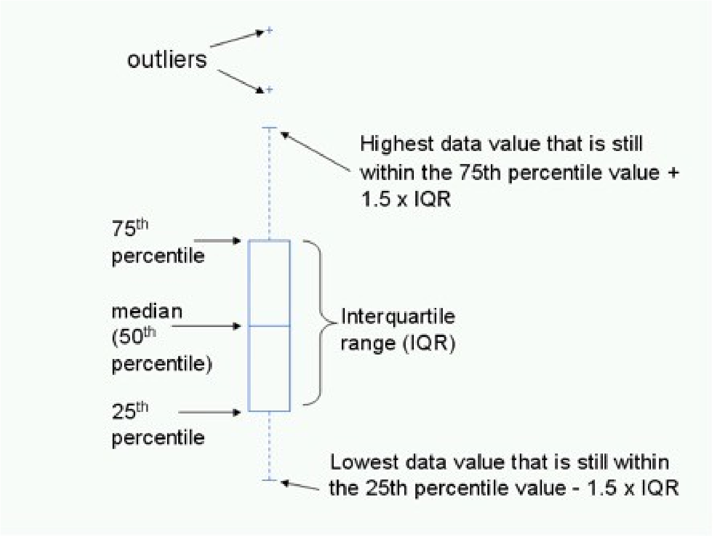
Using IQR Score

It is a measure of the dispersion similar to standard deviation or variance, but is much more robust against outliers.

In other words, the IQR is the first quartile subtracted from the third quartile; these quartiles can be clearly seen on a box plot on the data.

Algorithm for IQR

1. Sort the dataset in increasing order
2. Calculate the first quartile(q1) and third quartile (q3)
3. Find Interquartile range (q3-q1)
4. Find the lower bound - lower_bound = (q1 -1.5 * iqr)
5. Find the upper bound - upper_bound = (q3 +1.5 * iqr)
6. Anything that lies above or below the iqr is an outlier


outliers
Highest data value that is still within the 75th percentile value + 1.5 x IQR
75th percentile
median (50th percentile)
Interquartile range (IQR)
25th percentile
Lowest data value that is still within the 25th percentile value - 1.5 x IQR

```
#now we are going to fill the outlier values

numcols=data.dtypes[(data.dtypes=='float64')|(data.dtypes=='int64')].index

for x in numcols:
    descr=data[x].describe()
    IQR=descr['75%']-descr['25%']
    upper=descr['75%']+1.5*IQR
    lower=descr['25%']-1.5*IQR

    #replacing outlier values
    data[x][data[x]>upper]=upper
    data[x][data[x]<lower]=lower
```

Q19. Explain Correlation and multicollinearity?

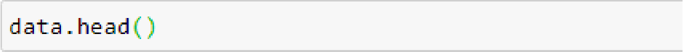
Correlation is a statistical measure that indicates the extent to which two or more variables move together[1]. A positive correlation indicates that the variables increase or decrease together. A negative correlation indicates that if one variable increases, the other decreases, and vice versa

Collinearity is a linear association between two predictors. Multicollinearity is a situation where two or more predictors are highly linearly related. In general, an absolute correlation coefficient of >0.7 among two or more predictors indicates the presence of multicollinearity.

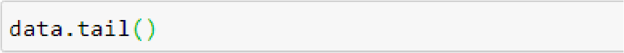
Q20. Explain any 5 pandas functions

1. Head and Tail

Once we read a dataset into a pandas data frame, we want to take a look at it to get an overview. The simplest way is to display some rows. Head and tail allow us to display rows from the top of bottom of data frame, respectively.



```
data.head()
```

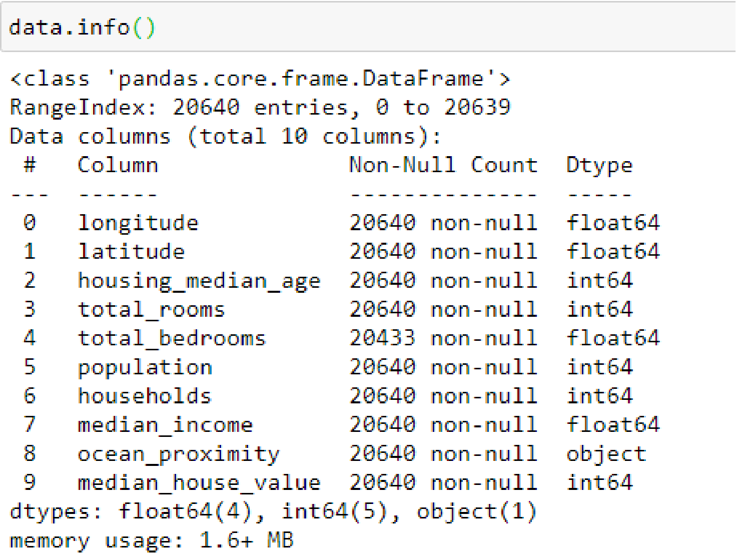


```
data.tail()
```

5 rows are displayed by default but we can adjust it just by passing the number of rows we'd like to display.

2. DataFrame.info( )

Pandas **`dataframe.info()`** function is used to get a concise summary of the dataframe. It comes really handy when doing exploratory analysis of the data. To get a quick overview of the dataset we use the `dataframe.info()` function.



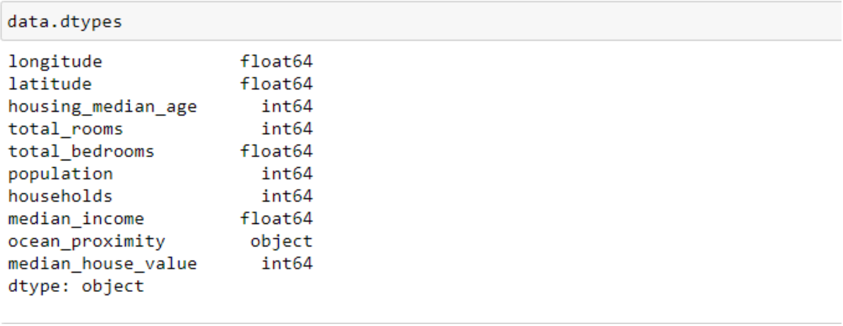
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  int64  
 3   total_rooms         20640 non-null  int64  
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  int64  
 6   households          20640 non-null  int64  
 7   median_income       20640 non-null  float64
 8   ocean_proximity     20640 non-null  object 
 9   median_house_value  20640 non-null  int64  
dtypes: float64(4), int64(5), object(1)
memory usage: 1.6+ MB
```

3. Dtypes

We need to have the values stored in an appropriate data type. Otherwise, we may encounter errors. For large datasets, memory-usage is greatly affected by correct data type selection. For example, "categorical" data type is more appropriate than "object" data type for categorical data especially when the number of categories is much less than the number of rows.

Dtypes shows the data type of each column.

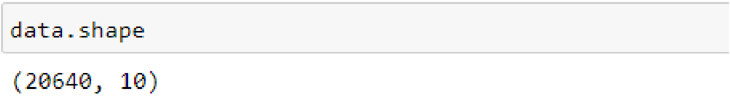


```
data.dtypes
```

```
longitude             float64
latitude              float64
housing_median_age      int64
total_rooms             int64
total_bedrooms        float64
population              int64
households              int64
median_income         float64
ocean_proximity        object
median_house_value      int64
dtype: object
```

4. Shape and Size

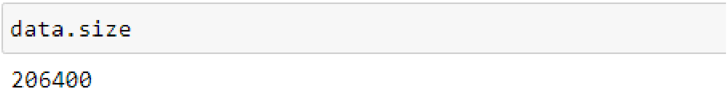
Shape can be used on numpy arrays, pandas series and dataframes. It shows the number of dimensions as well as the size in each dimension.



```
data.shape
```

```
(20640, 10)
```

Size, as the name suggests, returns the size of a dataframe which is the number of rows multiplied by the number of columns.



```
data.size
```

```
206400
```