**Pandas**

**pandas** is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python.

Moreover, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language.**

**Uses of Pandas for different Dataset:**

1. Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
2. Ordered and unordered (not necessarily fixed-frequency) time series data.
3. Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
4. Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

**Pandas data Structure:**

The two primary data structures of pandas, **Series** (1-dimensional) and **DataFrame** (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For **R** users, **DataFrame** provides everything that R’s **data.frame** provides and much more. Since pandas is built on top of  **Numpy**, it is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

**Below are just few things that pandas perform well**.

* Easy handling of **missing data** (represented as NaN) in floating point as well as non-floating point data.
* Size mutability: columns can be **inserted and deleted** from DataFrame and higher dimensional objects.
* Automatic and explicit **data alignment**: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let *Series*, *DataFrame*, etc. automatically align the data for you in computations.
* Powerful, flexible **group by** functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
* Make it **easy to convert** ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects.
* Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets.
* Intuitive **merging** and **joining** data sets.
* Flexible **reshaping** and pivoting of data sets.
* Robust IO tools for loading data from **flat files** (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast **HDF5 format**
* **Time series**-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging.

Now lets jump into questions which are being asked during technical interview.

1. **What is pandas and where it is used?**

Pandas is defined as an open-source library that provides high-performance data manipulation in Python. Since it provides fast, flexible and expressive data handling, it is mostly used in data analysis.

1. **What types of data Structure it provide?.**

Pandas offer two main types of data structure.

* Series (One- Dimensional)
* DataFrame (two-Dimensional)

1. **Explain Pandas’ Series.**

Series is a one-dimensional labelled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.).

Syntax:

Sr **= pd.Series(data =** None**, index =** None**, dtype =** None**, name =** None**, copy =** False**, fastpath =** False**)**

Example:

import pandas as pd

l1 = [10,20,30,40]

sr = pd.Series(data = l1,index = ['a','b','c','e'],dtype = float)

print(sr)

Output:

a 10.0

b 20.0

c 30.0

e 40.0

dtype: float64

1. **How to Create Series using Dictionary. How it works ?**

Series can be easily created by just passing dictionary to Series. When dictionary is passed but index not passed, Series make dictionary’ keys as series’ index and value as series value. If an index is passed, the values in data corresponding to the labels in the index will be pulled out.

**Example**:

dict = {'one':1,'two':2,'m':100}

sr = pd.Series(dict) # Without index

sr1 = pd.Series(dict, index = ['a','two','s']) # with Index

print("======Series without index=====")

print(sr)

print("======Series with index=====")

print(sr1) **Output**:

======Series without index=====

one 1

two 2

m 100

dtype: int64

======Series with index=====

a NaN #NaN because ‘a’ not defined in dict

two 2.0 # 2.0 because dictionary has value accociated with 2

s NaN # Same as a

dtype: float64

#Not: NaN (not a number) is the standard missing data marker used in pandas.

1. **What is DataFrame in pandas? How to create it?**

**DataFrame** is a 2-dimensional labelled data structure with columns of different types. We can think of DataFrame like a spreadsheet or SQL table, or a dictionary of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input:

* Dict of 1D ndarrays, lists, dicts, or Series.
* 2-D numpy. ndarray
* [Structured or record](https://docs.scipy.org/doc/numpy/user/basics.rec.html) ndarray
* A Series
* Another DataFrame

Syntax:

**dataFrame = pd.DataFrame(data =** None**, index =** None**, columns =** None**, dtype =** None**,copy =** False **)**

**Example:**

Import pandas as pd

sr\_name = pd.Series(['Roman','Ganga','Khan'])

sr\_salary = pd.Series([60000,80000,75000])

sr\_arrear = pd.Series([999,999,999])

dataSet = {'name' : sr\_name,'salary': sr\_salary,'arrear':sr\_arrear}

dataFrame = pd.DataFrame(dataSet)

dataFrame

**output**:

**name salary arrear**

0 Roman 60000 999

1 Ganga 80000 999

3 khan 75000 999

1. **How to do selection, addition and deletion of columns in pandas?**

We can treat a DataFrame semantically like a dict of like-indexed Series objects. Getting, setting, and deleting columns works with the same syntax as the dict operations:

**Selection of column/columns:**

dataFrame[‘name’] or dataFrame.name.

Multi-columns: dataFrame[[‘name’,salary’]]

**Addition**:

dataFrame['salary'] + dataFrame['arrear']

**Deletion**:

dataFrame.pop('arrear') # drop arrear column

1. **What is Reindexing in pandas?**

Pandas**dataframe.reindex()** function sets DataFrame row to new index with optional filling logic, placing NA/NaN in locations having no value in the previous index. A new object is produced unless the new index is equivalent to the current one and copy=False.

**Example**:

dataSet = {

'name' : ['Roman','Ganga','Khan'],

'salary': [60000,80000,75000],

'arrear':[999,999,999]}

dataFrame = pd.DataFrame(dataSet,index = ['first','second','third'])

dataFrame

**Output:**

**Name salary arrear**

**First** Roman 60000 999

**Second** Ganga 60000 999

**Third** khan 75000 999

**# Now apply reindexing**

dataFrame.reindex(['second','third','forth','first'])

**output**:

name salary arrear

second Ganga 60000 999

third khan 75000 999

forth NaN NaN NaN

first Roman 60000 999

1. **What is indexing in pandas?**

Indexing in pandas simply means selecting particular rows and columns of data from a DataFrame. Indexing could be selecting all the rows and some of the columns or 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**. we can perform indexing using below methods.

* **DataFrame[]** : used to select row and columns
* **DataFrame.loc[]:** This function is used for labels.
* **DataFrame.iloc[] :** This function is used for positions or integer based
* **DataFrame.ix[] :** This function is used for both label and integer based

1. **What is difference between loc and iloc?**

In pandas, **loc** is label based indexing, which means that we have to specify the name of the rows and columns that we need to filter out, whereas **iloc** is integer based indexing, means we have rows and columns by their integer index.

1. **How to convert DataFrame to Numpy array** ?

In order to convert DataFrame to Numpy array, we can use the property ‘**values’** of the dataFrame to convert **Dataframe** into **Numpy** array.

Syntax :

DataFrame.values.

1. **How to detect number of missing values in DataFrame per columns.?**

Since **isna()** function is used to detect missing values in DataFrmae, so we can sum all missing value with the help of i**sna()** function to get missing value per columns.

Example:

import pandas as pd

import numpy as np

dataSet1 = {

'name':['Roman','Setch','Ganga','Gold'],

'country':['America','Canada','India',np.nan],

'Salary':[15000,np.nan,18000,np.nan]

}

data = pd.DataFrame(dataSet1)

data.head()

**Output:**

Name country Salary

0 Roman America 15000

1 Seth Canada NaN

3 Ganga Unda 18000

4 Gold NaN NaN

Now Lets, find out missing values.

data.isna().sum()

**output**:

name 0

country 1

Salary 2

1. **How to add a new column at specific location in DataFrame?**

In order to insert a new column at specified location, we can use **insert()**  function.

This function insert column into DataFrame at specifies location.

Syntax:

DataFrame.insert(self,loc,columns, value[…..])

data.insert(2,'HRA', [10,20,30,50])

data

**output:**

name country HRA Salary

0 Roman America 10 15000

1 Seth Canada 20 NaN

2 Ganga India 30 18000

3 Gold NaN 50 NaN

1. **What is Difference Between appy() and applymap() in pandas?**

**Apply()** method can be applied for both Series and DataFrame elements where as **applymap ()** function only work on DataFrameElement. Both methods allow us to pass a function and apply it on every single element of Series / DataFrame.

**Example**:

import pandas as pd

dataset = {'salary':[20000,30000,40000],'PF':[2500,3200,1200]}

data1 = pd.DataFrame(dataset)

data1

**#Output:**

**Salary PF**

0 20000 2500

1 30000 3200

2 40000 1200

data1[['PF','salary']].apply(lambda x : x + 99)

**Output:**

**Salary PF**

0 20099 2599

1 30099 3299

2 40099 1299

data1.applymap(lambda x : x + 99)

**Output:**

**Salary PF**

0 20099 2599

1 30099 3299

2 40099 1299

1. **What nunique() function does in pandas ?**

Pandas **DataFrame.nunique()** function return a series with number of distinct/unique observation over the requested axis.

Example:

Import pandas as pd

dataSet1 = {

'name':['Roman','Setch','Roman','Gold'],

'country':['America','Canada','America','Japan'],

'Salary':[15000,20000,15000,13000]

}

data = pd.DataFrame(dataSet1)

data.head()

**Output:**

Name country Salary

0 Roman America 15000

1 Seth Canada 20000

3 Roman America 15000

4 Gold Japan 13000

data.nunique(axis = 0)

**Output**:

Name 3

Country 3

Salary 3

1. **What is time Series in Pandas ?**

A time series is an ordered sequence of data which basically represents how some quantity changes over time. pandas contains extensive capabilities and features for working with time series data for all domains.

**Pandas supports:**

* Parsing time series information from various sources and formats
* Generate sequences of fixed-frequency dates and time spans
* Manipulating and converting date time with timezone information
* Resampling or converting a time series to a particular frequency
* Performing date and time arithmetic with absolute or relative time increments

1. **Can Graphs be plotted using Pandas?**

Yes ,Graphs can be plotted using pandas . Pandas provides rich library to plot graph directly using dataFrame. There are some few functions listed below.

* DataFrame.plot()
* DataFrame.plot.area()
* DataFrame.plot.bar()
* DataFrame.plot.box()
* DataFrame.plot.hist()
* and many more

1. **Explain ‘describe()’ function of Pandas. Can we change default percentile of describe() function?**

Pandas’ **describe()** function is used to summarize some basic statistical detail of numeric values excluding NaN. However, it can also provide statistical detail of Categorical and object type of data if datatypes provided to it’s **include** parameter of this function. But, bydefault, it provides statistical summary of numerical values.

The default percentile of od describe function is [.25, .5, .75]. However, we can change the default percentile value of the summary function.

**Example**:

1. **What inplace parameter does in pandas?**

If we want to override existing DataFrame the, we have set **inplace=True** else **inplace = False.** However, default value of **inplace = False** in pandas.

**Example**:

import pandas as pd

import numpy as np

dataSet1 = {

'name':['Roman','Setch','Ganga',np.nan],

'country':['America','Canada','India','Japan'],

'Salary':[15000,12000,18000,np.nan],

}

data = pd.DataFrame(dataSet1)

data.head()

**Output:**

Name country Salary

0 Roman America 15000

1 Seth Canada 12000

2 Ganga India 18000

3 NaN Japan NaN

Now we are going to delete Nan rows with default value of **inplace** which is **False**

data.dropna(axis = 0,inplace = False) # drop NaN rows

Now Check DataFrame

Data.head()

**Output:**

Name country Salary

0 Roman America 15000

1 Seth Canada 12000

2 Ganga India 18000

3 NaN Japan NaN

We can see that Nan still exist in dataFrame. IF we want to store updated dataFrame, we have to assign it to another dataFrame.

Now drop **NaN** values by setting **inplace = True.**

data.dropna(axis = 0,inplace = True)

data.head()

**Output:**

Name country Salary

0 Roman America 15000

1 Seth Canada 12000

2 Ganga India 18000

1. **What is Categorical data in pandas?**

*Categoricals* are a pandas data type corresponding to categorical variables in statistics.

A categorical variable takes on a limited, and usually fixed, number of possible values.

**Examples** : gender, social class, blood type, country etc.

1. **How can we converts DataFrame to Numpy array ?**

Since Pandas is built on top of numpy, we can easyly convert dataFrame into numpy array.

We can use **DataFrame.to\_numpy()** function in order to convert dataFrame to numpy array