Experiment 12

Write python programs to implement Basic operations using pandas like series, data frames, indexing, filtering, combining and merging data frames etc.

Roll No.	61
Name	V Krishnasubramaniam
Class	D10-A
Subject	Python Lab
LO Mapped	LO1: Understand the structure, syntax, and semantics of the Python language LO6: Design and Develop cost-effective robust applications using the latest Python trends and technologies.

Aim:

Write python programs to implement Basic operations using pandas like series, data frames, indexing, filtering, combining and merging data frames etc.

Introduction:

Pandas

Pandas being one of the most popular packages in Python is widely used for data manipulation. It is a very powerful and versatile package which makes data cleaning and wrangling much easier and pleasant. The Pandas library has a great contribution to the python community and it makes python as one of the top programming languages for data science and analytics. It has become the first choice of data analysts and scientists for data analysis and manipulation.

Pandas package has many functions which are the essence for data handling and manipulation. In short, it can perform the following tasks for you:

1. Create a structured data set similar to R's data frame and Excel spreadsheet.
2. Reading data from various sources such as CSV, TXT, XLSX, SQL database, R etc.
3. Selecting particular rows or columns from the data set.
4. Arranging data in ascending or descending order.
5. Filtering data based on some conditions.
6. Summarizing data by classification variable.
7. Reshape data into wide or long format.
8. Time series analysis.
9. Merging and concatenating two datasets.
10. Iterate over the rows of the dataset.
11. Writing or Exporting data in CSV or Excel format.

Installation of Pandas

To install Pandas, below are the given steps to install Pandas in Python. You can use the pip command.

pip install pandas

To import the Pandas package, you can use the following command.

import pandas as pd

To import the dataset, you can use the function read_csv() to make it read a CSV file.

furniture = pd.read_csv(‘furniture.csv’)

Series

Technically, Pandas Series is a one-dimensional labeled array capable of holding any data type. In layman terms, Pandas Series is nothing but a column in an excel sheet. As depicted in the picture below, columns with Name, Age and Designation representing a Series.

Creating a series from List:

A Pandas Series can be created out of a Python list or NumPy array. It has to be remembered that unlike Python lists, a Series will always contain data of the same type. This makes the NumPy array a better candidate for creating a pandas series. Here is how we can use both of the above to create a Pandas Series

```
series_list = pd.Series([1,2,3,4,5,6])
series_np = pd.Series(np.array([10,20,30,40,50,60]))
```

Just as while creating the Pandas DataFrame, the Series also generates by default row index numbers which is a sequence of incremental numbers starting from '0'.

Creating a series from Dictionary:

As we've seen during creation of Pandas DataFrame, it was extremely easy to create a DataFrame out of python dictionaries as keys map to Column names while values correspond to list of column values. If we create a Series from a python dictionary, the key becomes the row index while the value becomes the value at that row index.

```
t_dict = {'a' : 1, 'b': 2, 'c':3}
series_dict = pd.Series(t_dict)
```

Series out of a Pandas DataFrame:

Though Pandas Series is extremely useful in itself for doing data analysis and provides many useful helper functions, most of the time, however, the analytic requirements will force us to use DataFrame and Series together.

```
my_dict = {
'name' : ["a", "b", "c", "d", "e"],
'age' : [10,20, 30, 40, 50],
'designation': ["CEO", "VP", "SVP", "AM", "DEV"]}
df = pd.DataFrame( my_dict,
index = [
"First -> ",
"Second -> ",
"Third -> ",
"Fourth -> ",
"Fifth -> "])
```

DataFrame provides two ways of accessing the column i.e by using dictionary syntax df['column_name'] or df.column_name. Each time we use these representations to get a column, we get a Pandas Series.

Series by iterating through columns of a DataFrame:

Pandas DataFrame is iterable and we can iterate through individual columns to get the series.

```
series_col = []
for col_name in df.columns:
        series_col.append(df[col_name])
```

Creating DataFrame using the Series (Standalone or combination):

A Pandas DataFrame is nothing but a collection of one of more Series (1+). We can generate the DataFrame by using a Single Series or by combining multiple Series.

```
df_from_series = pd.DataFrame([series_name, series_age])
```

Iterating over Series:

Just like many other data structures in python, it's possible to iterate over series using a simple for loop as:

```
for value in series_name:
        print(value)
```

We can also iterate over series row indexed as

```
for row_index in series_name.keys():
        print(row_index)
```

Dataframe

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

Dataframe data types:

Pandas Type	Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.

float64	float	Numeric characters with decimals. If a column contains numbers and NaNs, pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python standard library)	Values meant to hold time data. Look into these for time series experiments.

Dataframe attributes:

Df.attribute	**Description**
Dtypes	List the types of the columns
Columns	List the column names
Axes	List the row labels and column names
Ndim	Number of dimensions
Size	Number of elements
Shape	Return a tuple representing the dimensionality
Values	Numpy representation of the data

Dataframe methods:

Df.methods	**Description**
head([n]), tail([n])	First/last n rows
describe()	Generate descriptive statistics (for numeric columns only)
max(), min()	Return max/min values for all numeric columns
mean(), median()	Return mean/median values for all numeric columns
std()	Standard deviation
sample([n])	Returns a random sample of the data frame
dropna()	Drop all the records with missing values

Creating a Dataframe

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

DataFrame can be created using a single list or a list of lists.

```
lst = ['This', 'is', 'Python', 'Lab']
df = pd.DataFrame(lst)
print(df)
```

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

```
data = {'Name':['Tom',  'Jack'], 'Age':[20, 18]}
df = pd.DataFrame(data)
print(df)
```

Rows and Columns

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

Column Selection:

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

```
data = {'Name':['Jai', 'Anuj'],
        'Age':[27, 32],
        'Address':['Delhi', 'Kannauj'],
        'Qualification':['Msc', 'Phd']}
df = pd.DataFrame(data)
print(df[['Name', 'Qualification']])
```

Row Selection:

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

```
data = pd.read_csv("nba.csv", index_col ="Name")
first = data.loc["Avery Bradley"]
second = data.loc["R.J. Hunter"]
print(first, "\n\n\n", second)
```

Indexing and Selecting data

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

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

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

```
data = pd.read_csv("nba.csv", index_col ="Name")
first = data["Age"]
print(first)
```

Indexing a DataFrame using .loc[ ]:

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

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

```
data = pd.read_csv("nba.csv", index_col ="Name")
first = data.loc["Avery Bradley"]
second = data.loc["R.J. Hunter"]
print(first, "\n\n\n", second)
```

Indexing a DataFrame using .iloc[ ]:

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

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

```
data = pd.read_csv("nba.csv", index_col ="Name")
row2 = data.iloc[3]
print(row2)
```

Working with Missing Data

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

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

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

```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}
df = pd.DataFrame(dict)
df.isnull()
```

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

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

```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}
df = pd.DataFrame(dict)
df.fillna(0)
```

Dropping missing values using dropna():

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

```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, 40, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}
df = pd.DataFrame(dict)
df
```

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

```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, 40, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}
df = pd.DataFrame(dict)
df.dropna()
```

Iterating over Rows and Columns:

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

Iterating over rows:

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

```
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'score':[90, 40, 80, 98]}
df = pd.DataFrame(dict)
print(df)
```

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

```
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'score':[90, 40, 80, 98]}
df = pd.DataFrame(dict)
for i, j in df.iterrows():
            print(i, j)
            print()
```

Iterating over Columns:

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

```
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'score':[90, 40, 80, 98]}
df = pd.DataFrame(dict)
print(df)
```

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

```
columns = list(df)
for i in columns:
        print (df[i][2])
```

Filtering DataFrames

Pandas dataframe.filter() function is used to Subset rows or columns of dataframe according to labels in the specified index. Note that this routine does not filter a dataframe on its contents. The filter is applied to the labels of the index.

DataFrame.filter(items=None, like=None, regex=None, axis=None)

The items, like, and regex parameters are enforced to be mutually exclusive. axis defaults to the info axis that is used when indexing with []. Use filter() function to filter out any three columns of the dataframe.

```
df = pd.read_csv("nba.csv")
df.filter(["Name", "College", "Salary"])
```

Use filter() function to subset all columns in a dataframe which has the letter ‘a’ or ‘A’ in its name.
Use filter() function to subset all columns in a dataframe which has the letter ‘a’ or ‘A’ in its name.

```
df = pd.read_csv("nba.csv")
df.filter(regex ='[aA]')
```

The regular expression ‘[aA]’ looks for all column names which have an ‘a’ or an ‘A’ in its name.

Combining and merging DataFrames:

Pandas provides various facilities for easily combining together Series or DataFrame with various kinds of set logic for the indexes and relational algebra functionality in the case of Join/Merge-type operations. In addition, pandas also provides utilities to compare two Series or DataFrame and summarize their differences.

Concatenation:

The concat() function (in the main pandas namespace) does all of the heavy lifting of performing concatenation operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.

```
frames = [ process_your_file(f) for f in files ]
result = pd.concat(frames)
```

A useful shortcut to concat() are the append() instance methods on Series and DataFrame. These methods actually predated concat. They concatenate along axis=0, namely the index.

```
result = df1.append(df2)
```

For DataFrame objects which don’t have a meaningful index, you may wish to append them and ignore the fact that they may have overlapping indexes. To do this, use the ignore_index argument.

```
result = pd.concat([df1, df4], ignore_index=True, sort=False)
```

You can concatenate a mix of Series and DataFrame objects. The Series will be transformed to DataFrame with the column name as the name of the Series.

```
s1 = pd.Series(["X0", "X1", "X2", "X3"], name="X")
result = pd.concat([df1, s1], axis=1)
```

Merge:

Pandas provides a single function, merge(), as the entry point for all standard database join operations between DataFrame or named Series objects. Merge is a function in the pandas namespace, and it is also available as a DataFrame instance method merge(), with the calling DataFrame being implicitly considered the left object in the join.

result = pd.merge(left, right, how ='inner', on ='Key')

Users can use the validate argument to automatically check whether there are unexpected duplicates in their merge keys. Key uniqueness is checked before merge operations and so should protect against memory overflows. Checking key uniqueness is also a good way to ensure user data structures are as expected.

result = pd.merge(left, right, on="B", how="outer", validate="one_to_one")

merge() accepts the argument indicator. If True, a Categorical-type column called _merge will be added to the output object that takes on values.

pd.merge(df1, df2, on="col1", how="outer", indicator=True)

The indicator argument will also accept string arguments, in which case the indicator function will use the value of the passed string as the name for the indicator column. Merging will preserve the dtype of the join keys.

Join:

DataFrame.join() is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame.

result = left.join(right)

join() takes an optional argument which may be a column or multiple column names, which specifies that the passed DataFrame is to be aligned on that column in the DataFrame. These two function calls are completely equivalent.

left.join(right, on=key_or_keys)
pd.merge(left, right, left_on=key_or_keys, right_index=True, how="left", sort=False)

You can join a singly-indexed DataFrame with a level of a MultiIndexed DataFrame. The level will match on the name of the index of the singly-indexed frame against a level name of the MultiIndexed frame.

result = pd.merge(left.reset_index(), right.reset_index(), on=["key"], how="inner") .set_index(["key","Y"])

Strings passed as the on, left_on, and right_on parameters may refer to either column names or index level names. This enables merging DataFrame instances on a combination of index levels and columns without resetting indexes.

left_index = pd.Index(["K0", "K0", "K1", "K2"], name="key1")
right_index = pd.Index(["K0", "K1", "K2", "K2"], name="key1")

result = left.merge(right, on=["key1", "key2"])

A list or tuple of DataFrames can also be passed to join() to join them together on their indexes.

result = left.join([right, right2])

Another fairly common situation is to have two like-indexed (or similarly indexed) Series or DataFrame objects and wanting to "patch" values in one object from values for matching indices in the other.

df1 = pd.DataFrame([[np.nan, 3.0, 5.0], [-4.6, np.nan, np.nan], [np.nan, 7.0, np.nan]]
df2 = pd.DataFrame([[-42.6, np.nan, -8.2], [-5.0, 1.6, 4]], index=[1, 2])
result = df1.combine_first(df2)

A merge_ordered() function allows combining time series and other ordered data. In particular it has an optional fill_method keyword to fill/interpolate missing data.

pd.merge_ordered(left, right, fill_method="ffill", left_by="s")

A merge_asof() is similar to an ordered left-join except that we match on nearest key rather than equal keys. For each row in the left DataFrame, we select the last row in the right DataFrame whose on key is less than the left's key. Both DataFrames must be sorted by the key.

pd.merge_asof(trades, quotes, on="time", by="ticker")

Results:

Creating series from lists or dictionaries:

```
>>> import pandas as pd
>>> ser1 = pd.Series([1.5, 2.5, 3, 4.5, 5.0, 6])
>>> print(ser1)
0    1.5
1    2.5
2    3.0
3    4.5
4    5.0
5    6.0
dtype: float64
>>> ser3 = pd.Series(["A"]*4)
>>> print(ser3)
0    A
1    A
2    A
3    A
dtype: object
```

```
>>> ser4 = pd.Series({"India": "New Delhi","Japan": "Tokyo","UK": "London"})
>>> print(ser4)
India    New Delhi
Japan        Tokyo
UK          London
dtype: object
```

```
>>> ser5 = pd.Series({'D':10,'B':20,'C':30})
>>> print(ser5)
D    10
B    20
C    30
dtype: int64
```

```
>>> dictionary = {'A' : 50, 'B' : 10, 'C' : 80}
>>> ser6 = pd.Series(dictionary, index =['B', 'C', 'D', 'A'])
>>> print(ser6)
B    10.0
C    80.0
D     NaN
A    50.0
dtype: float64
```

Creating series from dataframes:

```
>>> data = {'First_Name': ['Jeff','Tina','Ben','Maria','Rob']}
>>> df = pd.DataFrame(data, columns = ['First_Name'])
>>> my_series = df.squeeze()
>>> print(my_series)
0     Jeff
1     Tina
2      Ben
3    Maria
4      Rob
Name: First_Name, dtype: object
>>> print (type(my_series))
<class 'pandas.core.series.Series'>
```

```
>>> data = {'First_Name': ['Jeff','Tina','Ben','Maria','Rob'],
...         'Last_Name':['Miller','Smith','Lee','Green','Carter'],
...         'Age':[33,42,29,28,57]
...         }
>>> df = pd.DataFrame(data, columns = ['First_Name','Last_Name','Age'])
>>> print(df)
  First_Name Last_Name  Age
0       Jeff    Miller   33
1       Tina     Smith   42
2        Ben       Lee   29
3      Maria     Green   28
4        Rob    Carter   57
>>> print (type(df))
<class 'pandas.core.frame.DataFrame'>
>>> my_series = df['Last_Name'].squeeze()
>>> print(my_series)
0    Miller
1     Smith
2       Lee
3     Green
4    Carter
Name: Last_Name, dtype: object
>>> print (type(my_series))
<class 'pandas.core.series.Series'>
```

```
>>> df = pd.DataFrame(data, columns = ['First_Name','Last_Name','Age'])
>>> my_series = df.iloc[3].squeeze()
>>> print(my_series)
First_Name    Maria
Last_Name     Green
Age              28
Name: 3, dtype: object
>>> print (type(my_series))
<class 'pandas.core.series.Series'>
```

Creating dataframe using series:

```
>>> author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']
>>> auth_series = pd.Series(author)
>>> print(auth_series)
0    Jitender
1     Purnima
2       Arpit
3       Jyoti
dtype: object
>>> print(type(auth_series))
<class 'pandas.core.series.Series'>
```

```
>>> author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']
>>> article = [210, 211, 114, 178]
>>> auth_series = pd.Series(author)
>>> article_series = pd.Series(article)
>>> frame = { 'Author': auth_series, 'Article': article_series }
>>> result = pd.DataFrame(frame)
>>> print(result)
     Author  Article
0  Jitender      210
1   Purnima      211
2     Arpit      114
3     Jyoti      178
```

```
>>> d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
...    'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
>>> df = pd.DataFrame(d)
>>> print(df)
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
```

```
>>> print ("Adding a new column by passing as Series:")
Adding a new column by passing as Series:
>>> df['three']=pd.Series([10,20,30],index=['a','b','c'])
>>> print(df)
   one  two  three
a  1.0    1   10.0
b  2.0    2   20.0
c  3.0    3   30.0
d  NaN    4    NaN
```

Iterating over series:

```
>>> sr = pd.Series([10, 25, 3, 25, 24, 6])
>>> index_ = ['Coca Cola', 'Sprite', 'Coke', 'Fanta', 'Dew', 'ThumbsUp']
>>> sr.index = index_
>>> print(sr)
Coca Cola    10
Sprite       25
Coke          3
Fanta        25
Dew          24
ThumbsUp      6
dtype: int64
>>> for items in sr.iteritems():
...     print(items)
...
('Coca Cola', 10)
('Sprite', 25)
('Coke', 3)
('Fanta', 25)
('Dew', 24)
('ThumbsUp', 6)
```

```
>>> sr = pd.Series([11, 21, 8, 18, 65, 84, 32, 10, 5, 24, 32])
>>> index_ = pd.date_range('2010-10-09', periods = 11, freq ='M')
>>> for items in sr.iteritems():
...     print(items)
...
(0, 11)
(1, 21)
(2, 8)
(3, 18)
(4, 65)
(5, 84)
(6, 32)
(7, 10)
(8, 5)
(9, 24)
(10, 32)
```

Creating Dataframes:

```
>>> cars = {'Brand': ['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],
...         'Price': [22000,25000,27000,35000]
...         }
>>> df = pd.DataFrame(cars, columns = ['Brand', 'Price'])
>>> df
            Brand  Price
0     Honda Civic  22000
1  Toyota Corolla  25000
2      Ford Focus  27000
3         Audi A4  35000
```

```
>>> df = pd.DataFrame(cars, columns = ['Brand','Price'], index=['Car_1','Car_2','Car_3','Car_4'])
>>> print (df)
               Brand  Price
Car_1     Honda Civic  22000
Car_2  Toyota Corolla  25000
Car_3      Ford Focus  27000
Car_4         Audi A4  35000
```

```
>>> data = {'Name':['Tom', 'nick', 'krish', 'jack'],
...         'Age':[20, 21, 19, 18]}
>>> df = pd.DataFrame(data)
>>> df
    Name  Age
0    Tom   20
1   nick   21
2  krish   19
3   jack   18
```

```
>>> data = {'Name':['Tom', 'Jack', 'nick', 'juli'],
...         'marks':[99, 98, 95, 90]}
>>> df = pd.DataFrame(data, index =['rank1',
...                                 'rank2',
...                                 'rank3',
...                                 'rank4'])
>>> df
       Name  marks
rank1   Tom     99
rank2  Jack     98
rank3  nick     95
rank4  juli     90
```

```
>>> data = [{'a': 1, 'b': 2, 'c':3},
...         {'a':10, 'b': 20, 'c': 30}]
>>> df = pd.DataFrame(data)
>>> df
    a   b   c
0   1   2   3
1  10  20  30
```

```
>>> data = [{'b': 2, 'c':3}, {'a': 10, 'b': 20, 'c': 30}]
>>> df = pd.DataFrame(data, index =['first', 'second'])
>>> df
         b   c     a
first    2   3   NaN
second  20  30  10.0
>>> data = [{'a': 1, 'b': 2},
...         {'a': 5, 'b': 10, 'c': 20}]
>>> df1 = pd.DataFrame(data, index =['first',
...                                   'second'],
...                     columns =['a', 'b'])
>>> df2 = pd.DataFrame(data, index =['first',
...                                   'second'],
...                     columns =['a', 'b1'])
>>> print (df1, "\n")
        a   b
first   1   2
second  5  10

>>> print (df2)
        a  b1
first   1 NaN
second  5 NaN
>>> Name = ['tom', 'krish', 'nick', 'juli']
>>> Age = [25, 30, 26, 22]
>>> list_of_tuples = list(zip(Name, Age))
>>> list_of_tuples
[('tom', 25), ('krish', 30), ('nick', 26), ('juli', 22)]
>>> df = pd.DataFrame(list_of_tuples,
...                   columns = ['Name', 'Age'])
>>> df
    Name  Age
0    tom   25
1  krish   30
2   nick   26
3   juli   22
```

```
>>> d = {'one' : pd.Series([10, 20, 30, 40],
...                        index =['a', 'b', 'c', 'd']),
...       'two' : pd.Series([10, 20, 30, 40],
...                         index =['a', 'b', 'c', 'd'])}
>>> df = pd.DataFrame(d)
>>> df
   one  two
a   10   10
b   20   20
c   30   30
d   40   40
```

Indexing and Selecting Data

```
>>> import numpy as np
>>> df = pd.DataFrame(np.random.randn(8, 4),
... index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C', 'D'])
>>> print(df.loc[:,'A'])
a    1.309302
b    1.365154
c    1.404699
d   -0.454994
e   -1.133364
f    1.025034
g    0.610755
h   -0.612311
Name: A, dtype: float64
```

```
>>> df = pd.DataFrame(np.random.randn(8, 4),
... index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C', 'D'])
>>> print(df.loc[:,['A','C']])
          A         C
a  1.898903 -0.186139
b  0.351128  1.123198
c -0.504657  1.299295
d  0.512774  0.738739
e  0.081164  0.140835
f  1.440522  0.608432
g -0.785954  1.061128
h -0.170317  0.475969
```

```
>>> df = pd.DataFrame(np.random.randn(8, 4),
... index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C', 'D'])
>>> print(df.loc[['a','b','f','h'],['A','C']])
          A         C
a  0.061635  0.011938
b  0.508719  2.757900
f -0.683978 -0.731219
h -0.764373  0.098186
```

```
>>> print(df.loc['a':'h'])
          A         B         C         D
a  0.061635 -0.704892  0.011938  0.532452
b  0.508719 -0.181856  2.757900  0.390956
c -0.794363  0.032345 -0.090895 -1.515903
d -2.734574 -0.334441  0.288970  1.221176
e  2.472113 -2.390828 -1.478872  0.210964
f -0.683978  0.670913 -0.731219  0.378647
g -0.105000 -0.007612 -0.084997  1.079773
h -0.764373  0.191552  0.098186 -0.833094

>>> print(df.loc['a']>0)
A     True
B    False
C     True
D     True
Name: a, dtype: bool


>>> df = pd.DataFrame(np.random.randn(8, 4), columns = ['A', 'B', 'C', 'D'])
>>> print(df.iloc[:4])
          A         B         C         D
0 -0.419160  0.338786 -0.702251 -0.271571
1 -1.243152  0.938335  1.149443  0.425362
2  1.563296 -1.140982 -1.187002  1.298654
3 -1.281237  2.569505 -0.611484  0.435721

>>> print(df.iloc[:4])
          A         B         C         D
0 -0.419160  0.338786 -0.702251 -0.271571
1 -1.243152  0.938335  1.149443  0.425362
2  1.563296 -1.140982 -1.187002  1.298654
3 -1.281237  2.569505 -0.611484  0.435721

>>> print(df.iloc[1:5, 2:4])
          C         D
1  1.149443  0.425362
2 -1.187002  1.298654
3 -0.611484  0.435721
4  0.022458 -1.081149

>>> print(df.iloc[[1, 3, 5], [1, 3]])
          B         D
1  0.938335  0.425362
3  2.569505  0.435721
5 -1.228962  0.719289
```

```
>>> print(df.iloc[1:3, :])
          A         B         C         D
1 -1.243152  0.938335  1.149443  0.425362
2  1.563296 -1.140982 -1.187002  1.298654

>>> print(df.iloc[:,1:3])
          B         C
0  0.338786 -0.702251
1  0.938335  1.149443
2 -1.140982 -1.187002
3  2.569505 -0.611484
4  0.151590  0.022458
5 -1.228962  1.530014
6 -0.193545  0.629984
7  1.437866 -0.786795

>>> print(df['A'])
0   -0.419160
1   -1.243152
2    1.563296
3   -1.281237
4    1.455593
5   -1.361588
6   -0.054949
7   -0.464120
Name: A, dtype: float64

>>> print(df[['A','B']])
          A         B
0 -0.419160  0.338786
1 -1.243152  0.938335
2  1.563296 -1.140982
3 -1.281237  2.569505
4  1.455593  0.151590
5 -1.361588 -1.228962
6 -0.054949 -0.193545
7 -0.464120  1.437866

>>> print(df[2:2])
Empty DataFrame
Columns: [A, B, C, D]
Index: []
```

```
>>> df.A
0   -0.419160
1   -1.243152
2    1.563296
3   -1.281237
4    1.455593
5   -1.361588
6   -0.054949
7   -0.464120
Name: A, dtype: float64
```

Missing Data:

```
>>> dict = {'First Score':[100, 90, np.nan, 95],
... 'Second Score': [30, 45, 56, np.nan],
... 'Third Score':[np.nan, 40, 80, 98]}
>>> df = pd.DataFrame(dict)
>>> df.isnull()
   First Score  Second Score  Third Score
0        False         False         True
1        False         False        False
2         True         False        False
3        False          True        False
```

```
>>> dict = {'First Score':[100, 90, np.nan, 95],
...         'Second Score': [30, 45, 56, np.nan],
...         'Third Score':[np.nan, 40, 80, 98]}
>>> df = pd.DataFrame(dict)
>>> df.notnull()
   First Score  Second Score  Third Score
0         True          True        False
1         True          True         True
2        False          True         True
3         True         False         True
```

```
>>> import pandas as pd
>>> import numpy as np
>>> dict = {'First Score':[100, 90, np.nan, 95],
...         'Second Score': [30, 45, 56, np.nan],
...         'Third Score':[np.nan, 40, 80, 98]}
>>> df = pd.DataFrame(dict)
>>> df.fillna(0)
   First Score  Second Score  Third Score
0        100.0          30.0          0.0
1         90.0          45.0         40.0
2          0.0          56.0         80.0
3         95.0           0.0         98.0
```

```
>>> dict = {'First Score':[100, 90, np.nan, 95],
...         'Second Score': [30, 45, 56, np.nan],
...         'Third Score':[np.nan, 40, 80, 98]}
>>> df = pd.DataFrame(dict)
>>> df.fillna(method ='bfill')
   First Score  Second Score  Third Score
0        100.0          30.0         40.0
1         90.0          45.0         40.0
2         95.0          56.0         80.0
3         95.0           NaN         98.0
```

```
>>> dict = {'First Score':[100, 90, np.nan, 95],
...         'Second Score': [30, np.nan, 45, 56],
...         'Third Score':[52, 40, 80, 98],
...         'Fourth Score':[np.nan, np.nan, np.nan, 65]}
>>> df = pd.DataFrame(dict)
>>> df.dropna()
   First Score  Second Score  Third Score  Fourth Score
3         95.0          56.0           98          65.0
```

Code in missing_data1.py

```
# importing pandas package
import pandas as pd
# making data frame from csv file
data = pd.read_csv("employees.csv")
# creating bool series True for NaN values
bool_series = pd.isnull(data["Gender"])
# displaying data only with Gender = NaN
print(data[bool_series])
```

Output:

```
C:\Users\vkris\Dropbox\Programming\Python\Experiments\Exp 12>py missing1.py
    First Name Gender  Start Date Last Login Time  Salary  Bonus % Senior Management                 Team
20        Lois    NaN   4/22/1995         7:18 PM   64714    4.934              True                Legal
22      Joshua    NaN    3/8/2012         1:58 AM   90816   18.816              True       Client Services
27       Scott    NaN   7/11/1991         6:58 PM  122367    5.218             False                Legal
31       Joyce    NaN   2/20/2005         2:40 PM   88657   12.752             False              Product
41   Christine    NaN   6/28/2015         1:08 AM   66582   11.308              True  Business Development
..         ...    ...         ...             ...     ...      ...               ...                  ...
961    Antonio    NaN   6/18/1989         9:37 PM  103050    3.050             False                Legal
972     Victor    NaN   7/28/2006         2:49 PM   76381   11.159              True                Sales
985    Stephen    NaN   7/10/1983         8:10 PM   85668    1.909             False                Legal
989     Justin    NaN   2/10/1991         4:58 PM   38344    3.794             False                Legal
995      Henry    NaN  11/23/2014         6:09 AM  132483   16.655             False          Distribution

[145 rows x 8 columns]
```

Code in missing_data2.py

```
# importing pandas package
import pandas as pd
# making data frame from csv file
data = pd.read_csv("employees.csv")
# creating bool series True for NaN values
bool_series = pd.notnull(data["Gender"])
# displayind data only with Gender = Not NaN
print(data[bool_series])
```

Output:

```
C:\Users\vkris\Dropbox\Programming\Python\Experiments\Exp 12>py missing2.py
    First Name  Gender Start Date Last Login Time  Salary  Bonus % Senior Management                 Team
0      Douglas    Male   8/6/1993        12:42 PM   97308    6.945              True            Marketing
1       Thomas    Male  3/31/1996         6:53 AM   61933    4.170              True                  NaN
2        Maria  Female  4/23/1993        11:17 AM  130590   11.858             False              Finance
3        Jerry    Male   3/4/2005         1:00 PM  138705    9.340              True              Finance
4        Larry    Male  1/24/1998         4:47 PM  101004    1.389              True       Client Services
..         ...     ...         ...             ...     ...      ...               ...                  ...
994     George    Male  6/21/2013         5:47 PM   98874    4.479              True            Marketing
996    Phillip    Male  1/31/1984         6:30 AM   42392   19.675             False              Finance
997    Russell    Male  5/20/2013        12:39 PM   96914    1.421             False              Product
998      Larry    Male  4/20/2013         4:45 PM   60500   11.985             False  Business Development
999     Albert    Male  5/15/2012         6:24 PM  129949   10.169              True                Sales

[855 rows x 8 columns]
```

Code in missing_data3.py

```
import pandas as pd
data = pd.read_csv("employees.csv")
data["Gender"].fillna("No Gender", inplace = True)
print(data)
```

Output:

```
C:\Users\vkris\Dropbox\Programming\Python\Experiments\Exp 12>py missing3.py
    First Name     Gender  Start Date Last Login Time  Salary  Bonus % Senior Management                Team
0      Douglas       Male    8/6/1993        12:42 PM   97308    6.945              True             Marketing
1       Thomas       Male   3/31/1996         6:53 AM   61933    4.170              True                   NaN
2        Maria     Female   4/23/1993        11:17 AM  130590   11.858             False               Finance
3        Jerry       Male    3/4/2005         1:00 PM  138705    9.340              True               Finance
4        Larry       Male   1/24/1998         4:47 PM  101004    1.389              True       Client Services
..         ...        ...         ...             ...     ...      ...               ...                   ...
995      Henry  No Gender  11/23/2014         6:09 AM  132483   16.655             False          Distribution
996    Phillip       Male   1/31/1984         6:30 AM   42392   19.675             False               Finance
997    Russell       Male   5/20/2013        12:39 PM   96914    1.421             False               Product
998      Larry       Male   4/20/2013         4:45 PM   60500   11.985             False  Business Development
999     Albert       Male   5/15/2012         6:24 PM  129949   10.169              True                 Sales

[1000 rows x 8 columns]
```

Code in missing_data4.py

```
import pandas as pd
data = pd.read_csv("employees.csv")
data["Gender"].fillna("No Gender", inplace = True)
print(data)
```

Output:

```
C:\Users\vkris\Dropbox\Programming\Python\Experiments\Exp 12>py missing4.py
    First Name  Gender  Start Date Last Login Time  Salary  Bonus % Senior Management                Team
0      Douglas    Male    8/6/1993        12:42 PM   97308    6.945              True             Marketing
1       Thomas    Male   3/31/1996         6:53 AM   61933    4.170              True                   NaN
2        Maria  Female   4/23/1993        11:17 AM  130590   11.858             False               Finance
3        Jerry    Male    3/4/2005         1:00 PM  138705    9.340              True               Finance
4        Larry    Male   1/24/1998         4:47 PM  101004    1.389              True       Client Services
..         ...     ...         ...             ...     ...      ...               ...                   ...
995      Henry     NaN  11/23/2014         6:09 AM  132483   16.655             False          Distribution
996    Phillip    Male   1/31/1984         6:30 AM   42392   19.675             False               Finance
997    Russell    Male   5/20/2013        12:39 PM   96914    1.421             False               Product
998      Larry    Male   4/20/2013         4:45 PM   60500   11.985             False  Business Development
999     Albert    Male   5/15/2012         6:24 PM  129949   10.169              True                 Sales

[1000 rows x 8 columns]
```

Code in missing_data5.py

```
import pandas as pd
data = pd.read_csv("employees.csv")
print(data[10:25])
```

Output:

```
C:\Users\vkris\Dropbox\Programming\Python\Experiments\Exp 12>py missing5.py
   First Name  Gender  Start Date Last Login Time  Salary  Bonus % Senior Management            Team
10     Louise  Female   8/12/1980         9:01 AM   63241   15.132              True             NaN
11      Julie  Female  10/26/1997         3:19 PM  102508   12.637              True           Legal
12    Brandon    Male   12/1/1980         1:08 AM  112807   17.492              True  Human Resources
13       Gary    Male   1/27/2008        11:40 PM  109831    5.831             False           Sales
14   Kimberly  Female   1/14/1999         7:13 AM   41426   14.543              True         Finance
15    Lillian  Female    6/5/2016         6:09 AM   59414    1.256             False         Product
16     Jeremy    Male   9/21/2010         5:56 AM   90370    7.369             False  Human Resources
17      Shawn    Male   12/7/1986         7:45 PM  111737    6.414             False         Product
18      Diana  Female  10/23/1981        10:27 AM  132940   19.082             False  Client Services
19      Donna  Female   7/22/2010         3:48 AM   81014    1.894             False         Product
20       Lois     NaN   4/22/1995         7:18 PM   64714    4.934              True           Legal
21    Matthew    Male    9/5/1995         2:12 AM  100612   13.645             False       Marketing
22     Joshua     NaN    3/8/2012         1:58 AM   90816   18.816              True  Client Services
23        NaN    Male   6/14/2012         4:19 PM  125792    5.042               NaN             NaN
24       John    Male    7/1/1992        10:08 PM   97950   13.873             False  Client Services
```

Filtering:

```
>>> record = {
...
...  'Name' : ['Ankit', 'Swapnil', 'Aishwarya',
...            'Priyanka', 'Shivangi', 'Shaurya' ],
...
...  'Age' : [22, 20, 21, 19, 18, 22],
...
...  'Stream' : ['Math', 'Commerce', 'Science',
...              'Math', 'Math', 'Science'],
...
...  'Percentage' : [90, 90, 96, 75, 70, 80] }
>>> dataframe = pd.DataFrame(record,
...                          columns = ['Name', 'Age',
...                                     'Stream', 'Percentage'])
```

```
>>> dataframe
        Name  Age    Stream  Percentage
0      Ankit   22      Math          90
1    Swapnil   20  Commerce          90
2  Aishwarya   21   Science          96
3   Priyanka   19      Math          75
4   Shivangi   18      Math          70
5    Shaurya   22   Science          80
>>> rslt_df = dataframe[dataframe['Percentage'] > 70]
>>> rslt_df
        Name  Age    Stream  Percentage
0      Ankit   22      Math          90
1    Swapnil   20  Commerce          90
2  Aishwarya   21   Science          96
3   Priyanka   19      Math          75
5    Shaurya   22   Science          80
>>> rslt_df = dataframe.loc[dataframe['Percentage'] > 70]
>>> rslt_df
        Name  Age    Stream  Percentage
0      Ankit   22      Math          90
1    Swapnil   20  Commerce          90
2  Aishwarya   21   Science          96
3   Priyanka   19      Math          75
5    Shaurya   22   Science          80
>>> options = ['Science', 'Commerce']
>>> rslt_df = dataframe[dataframe['Stream'].isin(options)]
>>> rslt_df
        Name  Age    Stream  Percentage
1    Swapnil   20  Commerce          90
2  Aishwarya   21   Science          96
5    Shaurya   22   Science          80
```

```
>>> rslt_df = dataframe.loc[dataframe['Stream'].isin(options)]
>>> rslt_df
        Name  Age    Stream  Percentage
1    Swapnil   20  Commerce          90
2  Aishwarya   21   Science          96
5    Shaurya   22   Science          80
>>> options = ['Commerce' ,'Science']
>>> rslt_df = dataframe[(dataframe['Age'] == 22) &
...           dataframe['Stream'].isin(options)]
>>> rslt_df
      Name  Age   Stream  Percentage
5  Shaurya   22  Science          80
>>> rslt_df = dataframe.loc[(dataframe['Age'] == 22) &
...               dataframe['Stream'].isin(options)]
>>> rslt_df
      Name  Age   Stream  Percentage
5  Shaurya   22  Science          80
```

Concatenation:

```
>>> df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
...                     'B': ['B0', 'B1', 'B2', 'B3'],
...                     'C': ['C0', 'C1', 'C2', 'C3'],
...                     'D': ['D0', 'D1', 'D2', 'D3']},
...                     index = [0, 1, 2, 3])
>>> df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
...                     'B': ['B4', 'B5', 'B6', 'B7'],
...                     'C': ['C4', 'C5', 'C6', 'C7'],
...                     'D': ['D4', 'D5', 'D6', 'D7']},
...                     index = [4, 5, 6, 7])
>>> pd.concat([df1,df2])
    A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
2  A2  B2  C2  D2
3  A3  B3  C3  D3
4  A4  B4  C4  D4
5  A5  B5  C5  D5
6  A6  B6  C6  D6
7  A7  B7  C7  D7
```

```
>>> one = pd.DataFrame({
...    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
...    'subject_id':['sub1','sub2','sub4','sub6','sub5'],
...    'Marks_scored':[98,90,87,69,78]},
...    index=[1,2,3,4,5])
>>> two = pd.DataFrame({
...    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
...    'subject_id':['sub2','sub4','sub3','sub6','sub5'],
...    'Marks_scored':[89,80,79,97,88]},
...    index=[1,2,3,4,5])
>>> pd.concat([one,two])
     Name subject_id  Marks_scored
1    Alex       sub1            98
2     Amy       sub2            90
3   Allen       sub4            87
4   Alice       sub6            69
5  Ayoung       sub5            78
1   Billy       sub2            89
2   Brian       sub4            80
3    Bran       sub3            79
4   Bryce       sub6            97
5   Betty       sub5            88
```

Merge:

```
>>> left = pd.DataFrame({'Key': ['K0', 'K1', 'K2', 'K3'],
...                      'A': ['A0', 'A1', 'A2', 'A3'],
...                      'B': ['B0', 'B1', 'B2', 'B3']})
>>> right = pd.DataFrame({'Key': ['K0', 'K1', 'K2', 'K3'],
...                       'C': ['C0', 'C1', 'C2', 'C3'],
...                       'D': ['D0', 'D1', 'D2', 'D3']})
>>> pd.merge(left, right, how ='inner', on ='Key')
  Key   A   B   C   D
0  K0  A0  B0  C0  D0
1  K1  A1  B1  C1  D1
2  K2  A2  B2  C2  D2
3  K3  A3  B3  C3  D3
```

```
>>> left = pd.DataFrame({
...    'id':[1,2,3,4,5],
...    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
...    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
>>> right = pd.DataFrame({
... 'id':[1,2,3,4,5],
...    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
...    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
>>> pd.merge(left,right,on='id')
   id  Name_x subject_id_x Name_y subject_id_y
0   1    Alex         sub1  Billy         sub2
1   2     Amy         sub2  Brian         sub4
2   3   Allen         sub4   Bran         sub3
3   4   Alice         sub6  Bryce         sub6
4   5  Ayoung         sub5  Betty         sub5
```

Join:

```
>>> left = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
...                      'B': ['B0', 'B1', 'B2', 'B3']},
...                      index = ['K0', 'K1', 'K2', 'K3'])
>>> right = pd.DataFrame({'C': ['C0', 'C1', 'C2', 'C3'],
...                       'D': ['D0', 'D1', 'D2', 'D3']},
...                       index = ['K0', 'K1', 'K2', 'K3'])
>>> left.join(right)
     A   B   C   D
K0  A0  B0  C0  D0
K1  A1  B1  C1  D1
K2  A2  B2  C2  D2
K3  A3  B3  C3  D3
```

```
>>> df = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3', 'K4', 'K5'],
...                    'A': ['A0', 'A1', 'A2', 'A3', 'A4', 'A5']})
>>> other = pd.DataFrame({'key': ['K0', 'K1', 'K2'],
...                       'B': ['B0', 'B1', 'B2']})
>>> df.join(other, lsuffix='_caller', rsuffix='_other')
  key_caller   A key_other    B
0         K0  A0        K0   B0
1         K1  A1        K1   B1
2         K2  A2        K2   B2
3         K3  A3       NaN  NaN
4         K4  A4       NaN  NaN
5         K5  A5       NaN  NaN
```

Conclusion:

Thus, we have understood the basics of data manipulation in Python using pandas. We learnt various ways to create, filter, merge, concatenate, and manipulate missing data in a Series and a Dataframe, by performing hands-on practical programs for each.