**Experiment No.1**

**Aim:** To understand the process of data preparation using NumPy and Pandas

**CO Mapped:**

**CO1:** To apply the process of data preparation for the given dataset to solve real-world problems

**Prerequisites:** Python3, basic syntax of NumPy and Pandas

**Theory:**

Data preparation is the process of preparing raw data so that it is suitable for further processing and analysis. Key steps include collecting, cleaning, and labelling raw data into a form suitable for machine learning (ML) algorithms and then exploring and visualizing the data.

**Derive an index field and add it to the data set**

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python packages. Pandas is one of those packages and makes importing and analysing data much easier.

Pandas set\_index() is a method to set a List, Series or Data frame as index of a Data Frame. Index column can be set while making a data frame too. But sometimes a data frame is made out of two or more data frames and hence later index can be changed using this method.  
Syntax: 

DataFrame.set\_index(keys, drop=True, append=False, inplace=False, verify\_integrity=False)

Parameters: 

keys: Column name or list of column name.   
drop: Boolean value which drops the column used for index if True.   
append: Appends the column to existing index column if True.   
inplace: Makes the changes in the dataframe if True.   
verify\_integrity: Checks the new index column for duplicates if True. 

Code #1: Changing Index column   
In this example, First Name column has been made the index column of Data Frame. 

|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("employees.csv")    # setting first name as index column  data.set\_index("First Name", inplace = True)    # display  data.head() |

Output:   
As shown in the output images, earlier the index column was a series of number but later it has been replaced with First name.  
Before operation – 



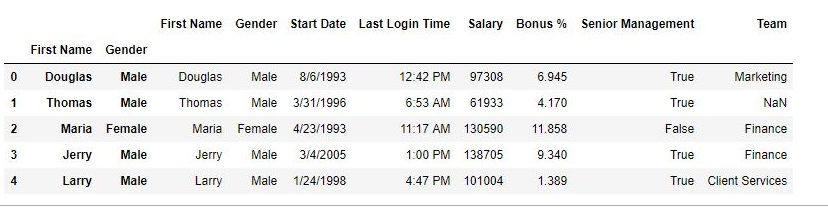
After operation – 



Code #2: Multiple index Column   
In this example, two columns will be made as index column. Drop parameter is used to Drop the column and append parameter is used to append passed columns to the already existing index column. 

|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("employees.csv")    # setting first name as index column  data.set\_index(["First Name", "Gender"], inplace = True,                              append = True, drop = False)    # display  data.head() |

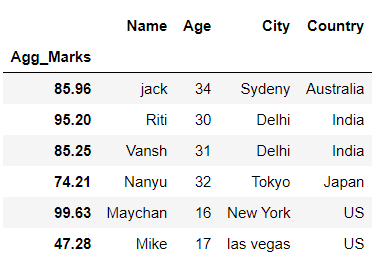
Output:   
As shown in the output Image, the data is having 3 index columns. 



Code #3: Setting a single Float column as Index in Pandas DataFrame

|  |
| --- |
| # importing pandas library  import pandas as pd    # creating and initializing a nested list  students = [['jack', 34, 'Sydeny', 'Australia',85.96],              ['Riti', 30, 'Delhi', 'India',95.20],              ['Vansh', 31, 'Delhi', 'India',85.25],              ['Nanyu', 32, 'Tokyo', 'Japan',74.21],              ['Maychan', 16, 'New York', 'US',99.63],              ['Mike', 17, 'las vegas', 'US',47.28]]    # Create a DataFrame object  df = pd.DataFrame(students,                        columns=['Name', 'Age', 'City', 'Country','Agg\_Marks'],                             index=['a', 'b', 'c', 'd', 'e', 'f'])    # here we set Float column 'Agg\_Marks' as index of data frame  # using dataframe.set\_index() function  df = df.set\_index('Agg\_Marks')      # Displaying the Data frame  df |

Output :

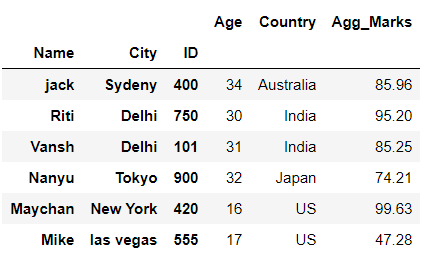


In the above example, we set the column ‘Agg\_Marks‘ as an index of the data frame.

Code #4: Setting three columns as MultiIndex in Pandas DataFrame

|  |
| --- |
| # importing pandas library  import pandas as pd    # creating and initializing a nested list  students = [['jack', 34, 'Sydeny', 'Australia',85.96,400],              ['Riti', 30, 'Delhi', 'India',95.20,750],              ['Vansh', 31, 'Delhi', 'India',85.25,101],              ['Nanyu', 32, 'Tokyo', 'Japan',74.21,900],              ['Maychan', 16, 'New York', 'US',99.63,420],              ['Mike', 17, 'las vegas', 'US',47.28,555]]    # Create a DataFrame object  df = pd.DataFrame(students,                        columns=['Name', 'Age', 'City', 'Country','Agg\_Marks','ID'],                             index=['a', 'b', 'c', 'd', 'e', 'f'])    # Here we pass list of 3 columns i.e 'Name', 'City' and 'ID'  # to dataframe.set\_index() function  # to set them as multiIndex of dataframe  df = df.set\_index(['Name','City','ID'])      # Displaying the Data frame  df |

Output :



In the above example, we set the columns ‘Name‘, ‘City‘, and ‘ID‘ as multiIndex of the data frame.

**Find out the missing values**

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 a real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, Suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing.  In Pandas missing data is represented by two value:

* None: None is a Python singleton object that is often used for missing data in Python code.
* NaN : NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

* isnull()
* notnull()
* dropna()
* fillna()
* replace()
* interpolate()

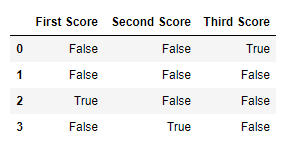
### Checking for missing values using isnull() and notnull()

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

#### Checking for missing values using isnull()

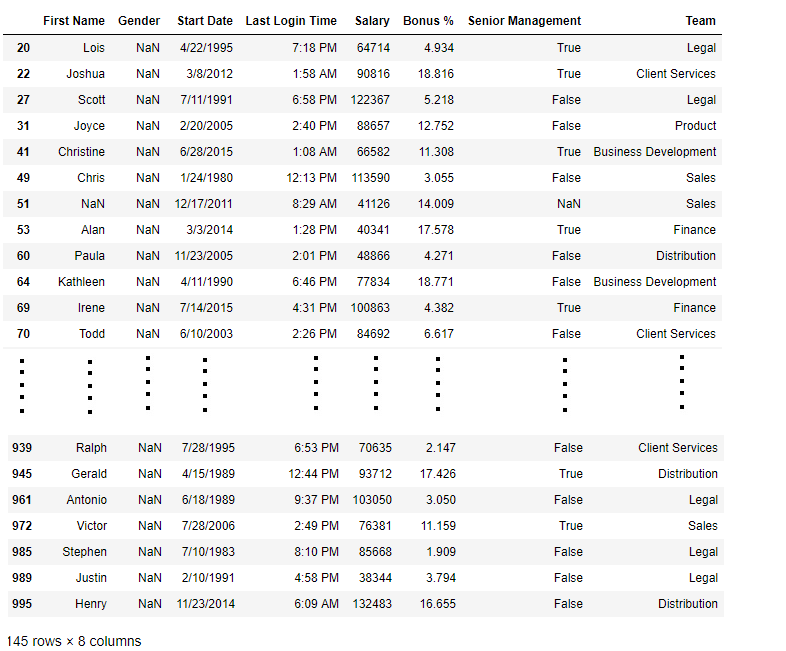
In order to check null values in Pandas DataFrame, we use isnull() function this function return dataframe of Boolean values which are True for NaN values. **Code #1:**

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, 45, 56, np.nan],          'Third Score':[np.nan, 40, 80, 98]}    # creating a dataframe from list  df = pd.DataFrame(dict)    # using isnull() function  df.isnull() |

**Output:**    **Code #2:**

* Python

|  |
| --- |
| # 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"])    # filtering data  # displaying data only with Gender = NaN  data[bool\_series] |

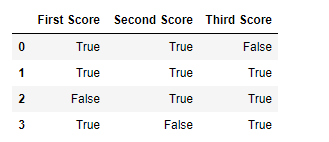
**Output:** As shown in the output image, only the rows having Gender = NULL are displayed. 

#### Checking for missing values using notnull()

In order to check null values in Pandas Dataframe, we use notnull() function this function return dataframe of Boolean values which are False for NaN values.

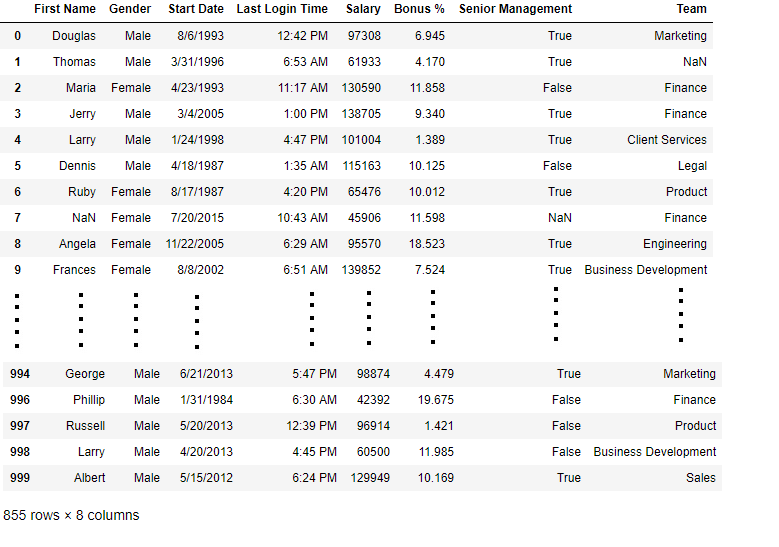
**Code #3:**

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, 45, 56, np.nan],          'Third Score':[np.nan, 40, 80, 98]}    # creating a dataframe using dictionary  df = pd.DataFrame(dict)    # using notnull() function  df.notnull() |

**Output:**    **Code #4:**

* Python

|  |
| --- |
| # 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"])    # filtering data  # displaying data only with Gender = Not NaN  data[bool\_series] |

**Output:** As shown in the output image, only the rows having Gender = NOT NULL are displayed. 

## **Finding outliers using statistical methods**

## Since the data doesn’t follow a normal distribution, we will calculate the outlier data points using the statistical method called interquartile range (IQR) instead of using Z-score. Using the IQR, the outlier data points are the ones falling below Q1–1.5 IQR or above Q3 + 1.5 IQR. The Q1 is the 25th percentile and Q3 is the 75th percentile of the dataset, and IQR represents the interquartile range calculated by Q3 minus Q1 (Q3–Q1).

Using the convenient pandas [.quantile()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.quantile.html) function, we can create a simple Python function that takes in our column from the dataframe and outputs the outliers:

#create a function to find outliers using IQR

def find\_outliers\_IQR(df):

   q1=df.quantile(0.25)

   q3=df.quantile(0.75)

   IQR=q3-q1

   outliers = df[((df<(q1-1.5\*IQR)) | (df>(q3+1.5\*IQR)))]

   return outliers

Notice using .[quantile()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.quantile.html) we can define Q1 and Q3. Next we calculate IQR, then we use the values to find the outliers in the dataframe. Since it takes a dataframe, we can input one or multiple columns at a time.

First run fare\_amount through the function to return a series of the outliers.

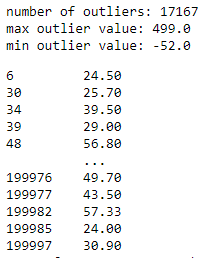
outliers = find\_outliers\_IQR(df[“fare\_amount”])

print(“number of outliers: “+ str(len(outliers)))

print(“max outlier value: “+ str(outliers.max()))

print(“min outlier value: “+ str(outliers.min()))

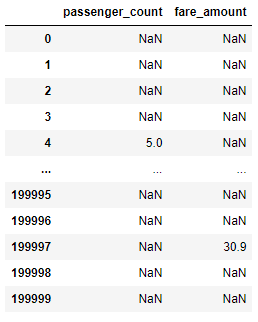
outliers validating the find\_outliers\_IQR function



Using the IQR method, we find 17,167 fare\_amount outliers in the dataset. I printed the min and max values to verify they match the statistics we saw when using the pandas describe() function, which helps confirm we calculated the outliers correctly.

We can also pass both fare\_amount and passenger\_count through the function to get back a dataframe of all rows instead of just the outliers. If the value is not an outlier, it will display as NaN (not a number):

outliers = find\_outliers\_IQR(df[[“passenger\_count”,”fare\_amount”]])

outliers find outliers IQR dataframe

**Conclusion**: -

In this experiment, we studied how using pandas and NumPy library we can pre-process the data.