Introduction to pandas Data Structures

Data Frame, Data Frame attributes and Methods

Summarizing and Computing Descriptive Statistics

#Introduction to Pandas, Numpy and other libraries in Python Loading data Splitting the dataset into training & testing data Exploring the functions of the pandas library - head(), read \_ csv(), tail(), shape(), describe(), dtypes() etc

## What is Pandas?

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

## Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

## What Can Pandas Do?

Pandas gives you answers about the data. Like:

* Is there a correlation between two or more columns?
* What is average value?
* Max value?
* Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called *cleaning* the data.

## Installation of Pandas

If you have [Python](https://www.w3schools.com/python/default.asp) and [PIP](https://www.w3schools.com/python/python_pip.asp) already installed on a system, then installation of Pandas is very easy.

Install it using this command:

C:\Users\*Your Name*>pip install pandas

## Import Pandas

Once Pandas is installed, import it in your applications by adding the import keyword:

Now Pandas is imported and ready to use.

### Example

import pandas

mydataset = {

'cars': ["BMW", "Volvo", "Ford"],

'passings': [3, 7, 2]

}

myvar = pandas.DataFrame(mydataset)

print(myvar)

## What is a Series?

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.

### Example

Create a simple Pandas Series from a list:

import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)

## Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

This label can be used to access a specified value.

### Example

Return the first value of the Series:

print(myvar[0])

## Create Labels

With the index argument, you can name your own labels.

### Example

Create your own labels:

import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

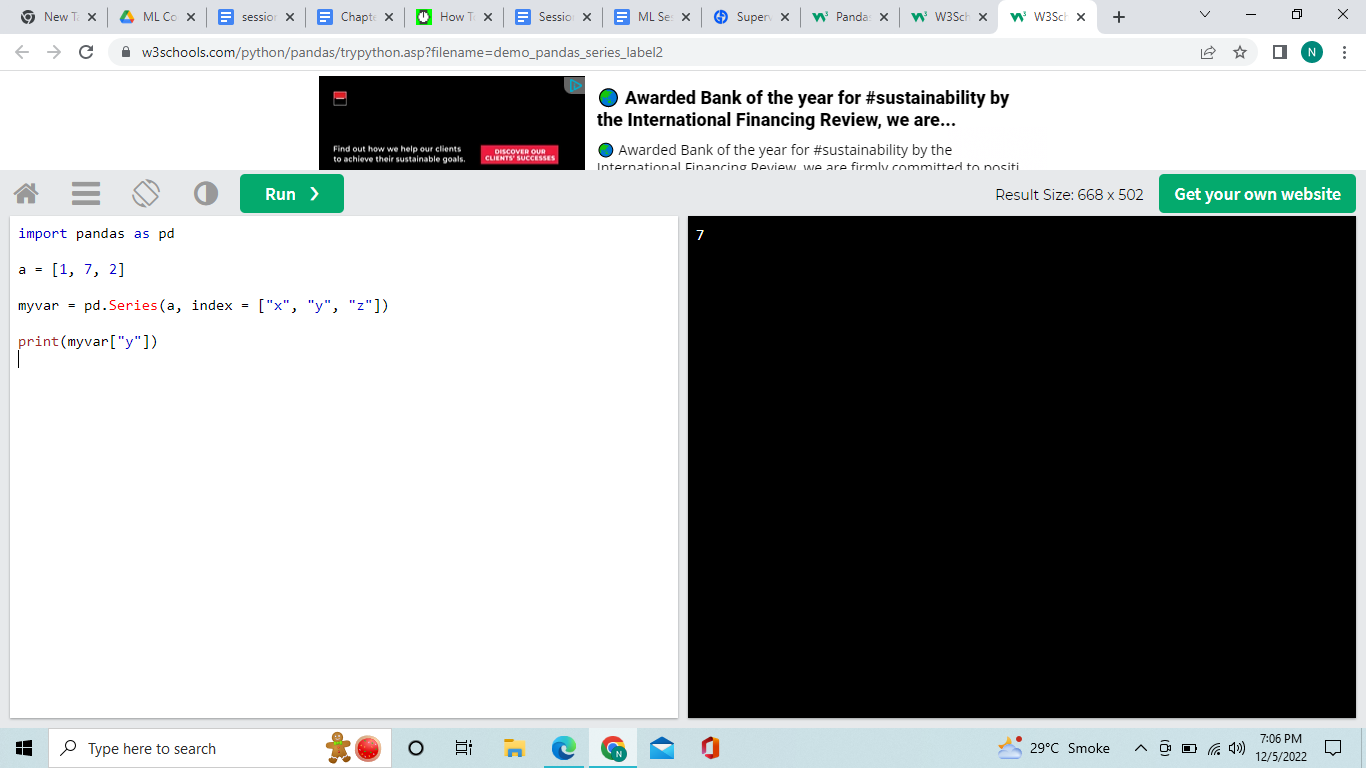
print(myvar)

Ex:

import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar["y"])

## Key/Value Objects as Series

You can also use a key/value object, like a dictionary, when creating a Series.

### Example

Create a simple Pandas Series from a dictionary:

import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)

## DataFrames

Data sets in Pandas are usually multi-dimensional tables, called DataFrames.

Series is like a column, a DataFrame is the whole table.

### Example

Create a DataFrame from two Series:

import pandas as pd

data = {

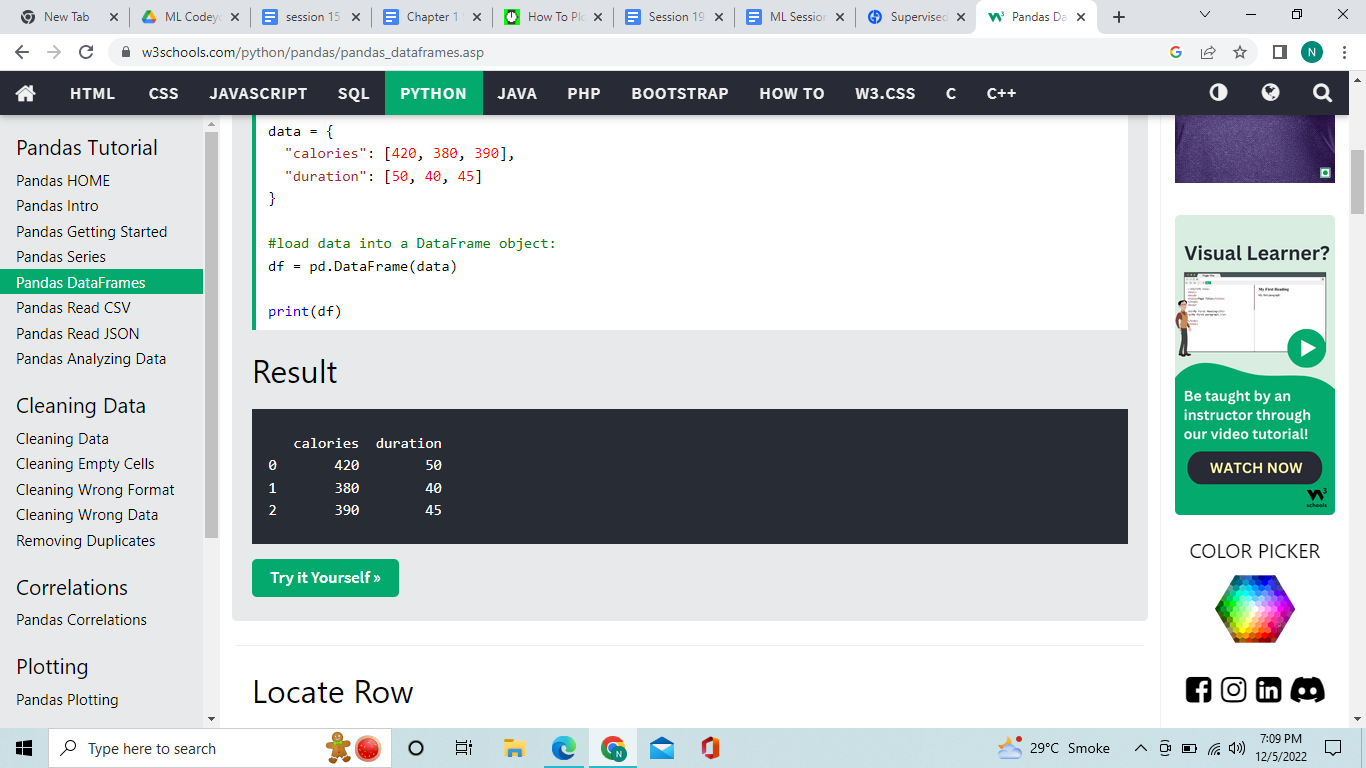
"calories": [420, 380, 390],

"duration": [50, 40, 45]

}

myvar = pd.DataFrame(data)

print(myvar)



## Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

### Example

Return row 0:

#refer to the row index:

print(df.loc[0])

## Named Indexes

## With the index argument, you can name your own indexes.

### Example

## Add a list of names to give each row a name:

## import pandas as pd

## data = {

## "calories": [420, 380, 390],

## "duration": [50, 40, 45]

## }

## df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

## print(df)

**Append:**

mydataset1={"cars":["BMW","Volvo","Ford"],"passings":[3,7,8]}

d=pd.DataFrame(mydataset1)

mydataset2={"cars":["Vv","pp"],"passings":[9,2]}

d2=pd.DataFrame(mydataset2)

df6=d.append(d2)

print(df6)

Output: cars passings

0 BMW 3

1 Volvo 7

2 Ford 8

0 Vv 9

1 pp 2

**Append:**

# Importing pandas as pd

import pandas as pd

# Creating the first Dataframe using dictionary

df1 = df = pd.DataFrame({"a":[1, 2, 3, 4],

"b":[5, 6, 7, 8]})

# Creating the Second Dataframe using dictionary

df2 = pd.DataFrame({"a":[1, 2, 3],

"b":[5, 6, 7]})

# Print df1

print(df1, "\n")

df1.append(df2,ignore\_index=True)# Ignore index will remove repetitive sequence of index.

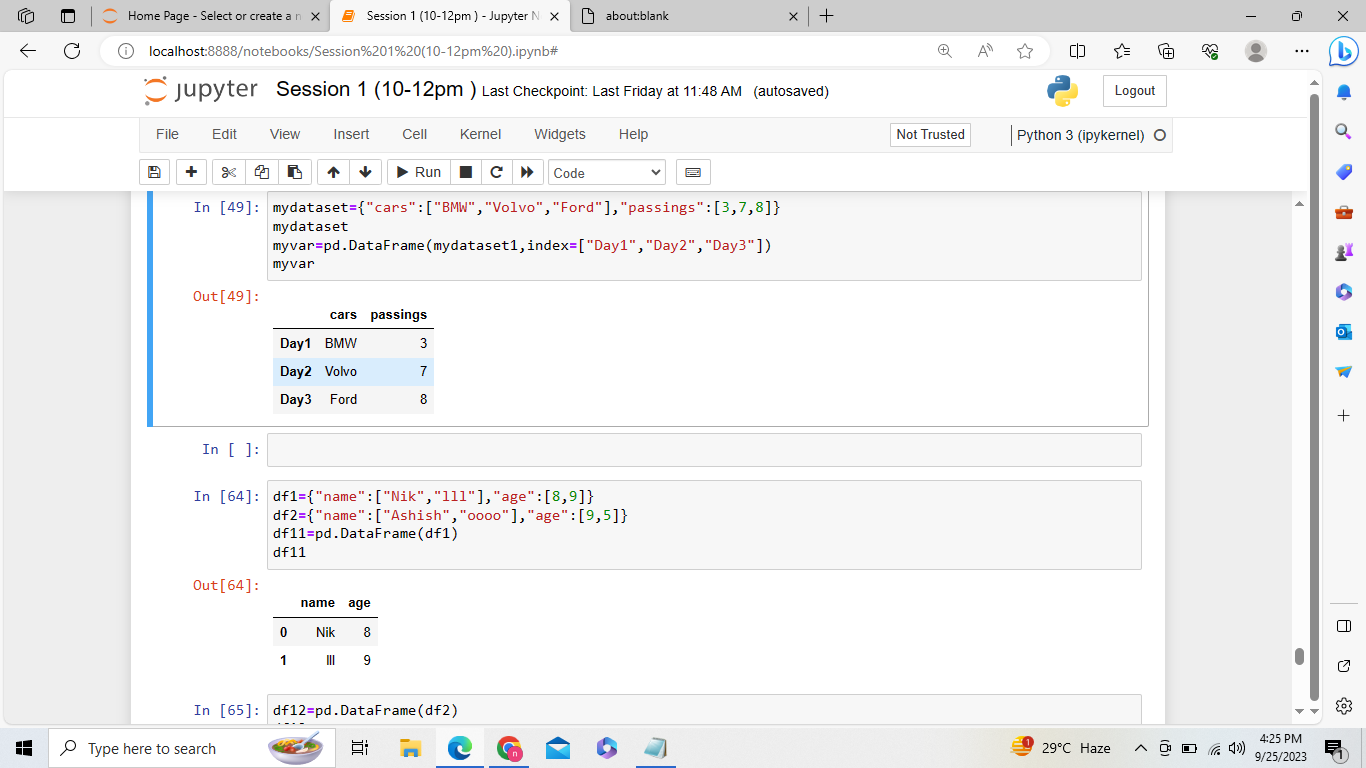
**Add index value**:mydataset={"cars":["BMW","Volvo","Ford"],"passings":[3,7,8]}

mydataset

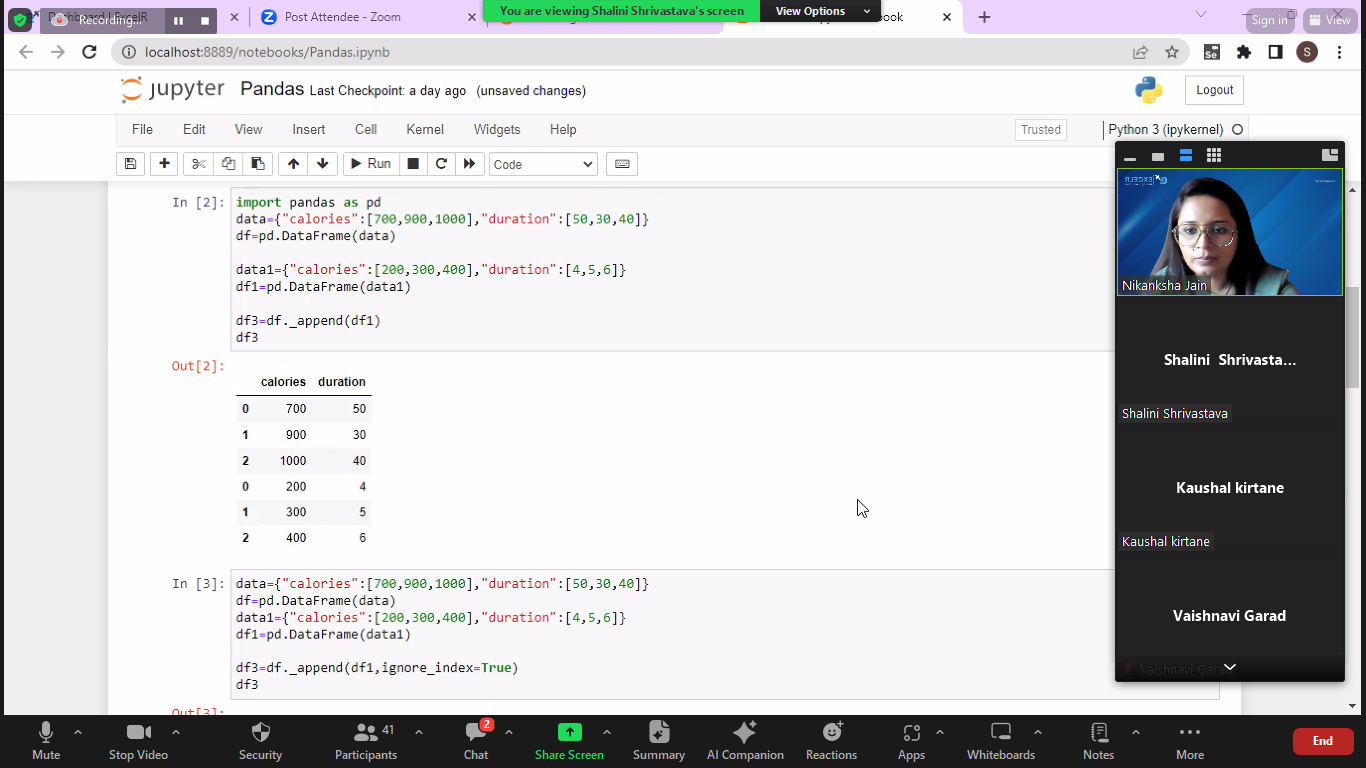
myvar=pd.DataFrame(mydataset1,index=["Day1","Day2","Day3"])

Myvar

Output:



Another append function:



**Concatnate:**

df1={"name":["Nik","lll"],"age":[8,9]}

df2={"name":["Ashish","oooo"],"age":[9,5]}

df11=pd.DataFrame(df1)

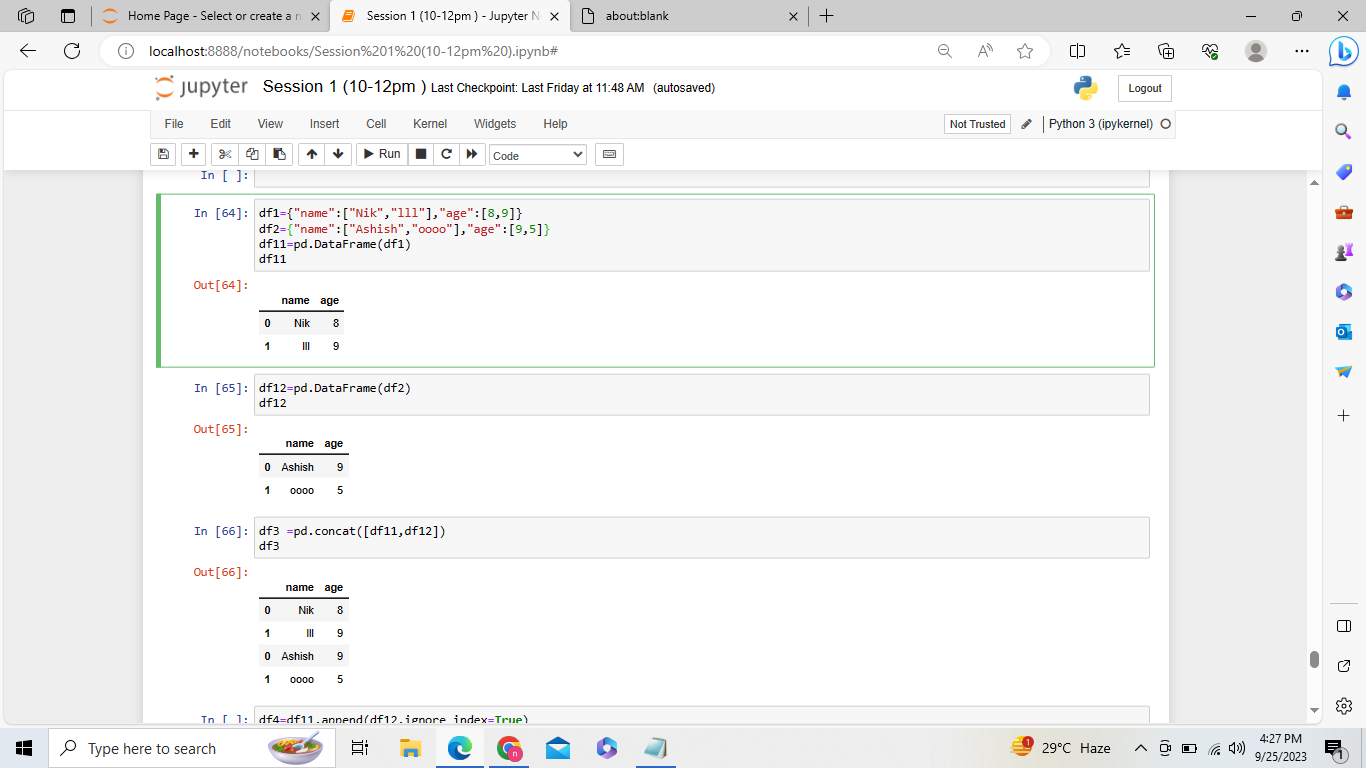
df11

df12=pd.DataFrame(df2)

df12

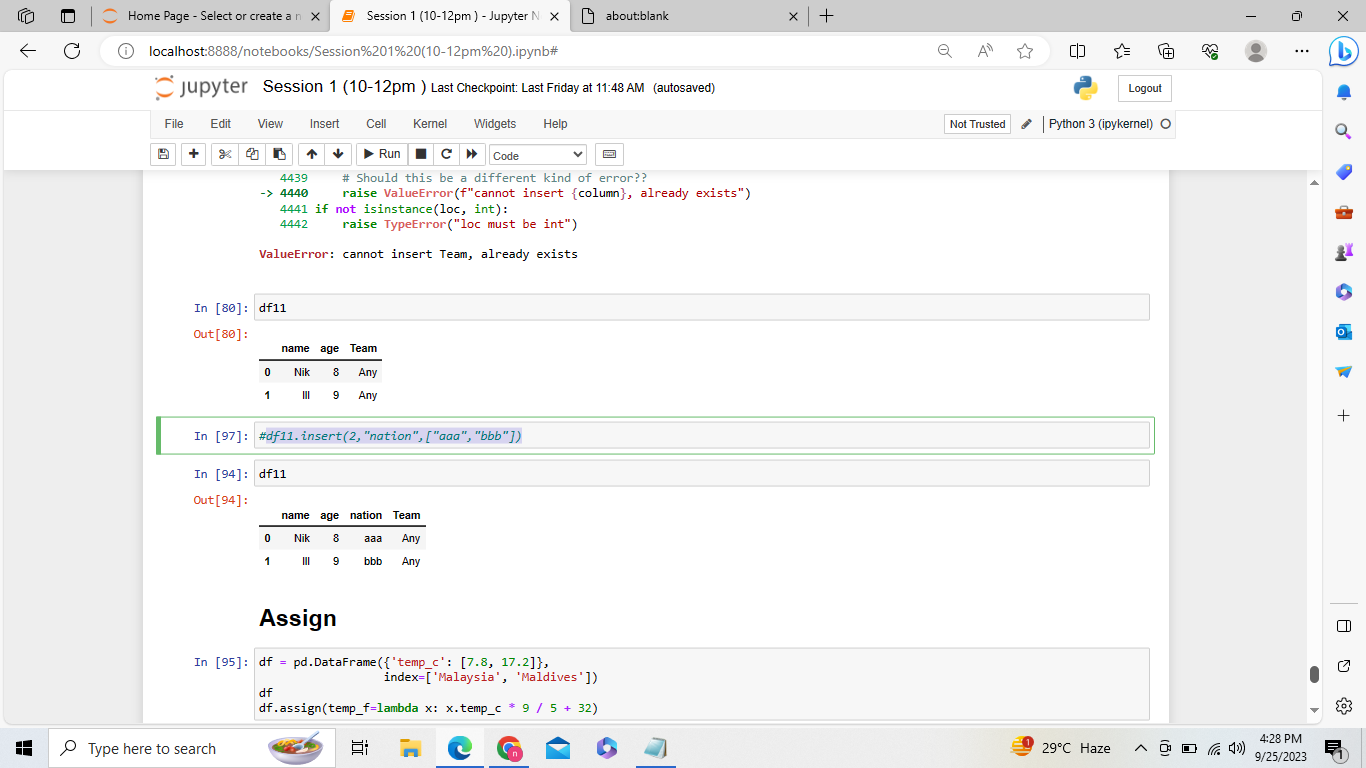
df3 =pd.concat([df11,df12])

df3



Insert:

df11.insert(2,"nation",["aaa","bbb"])



## Assign:

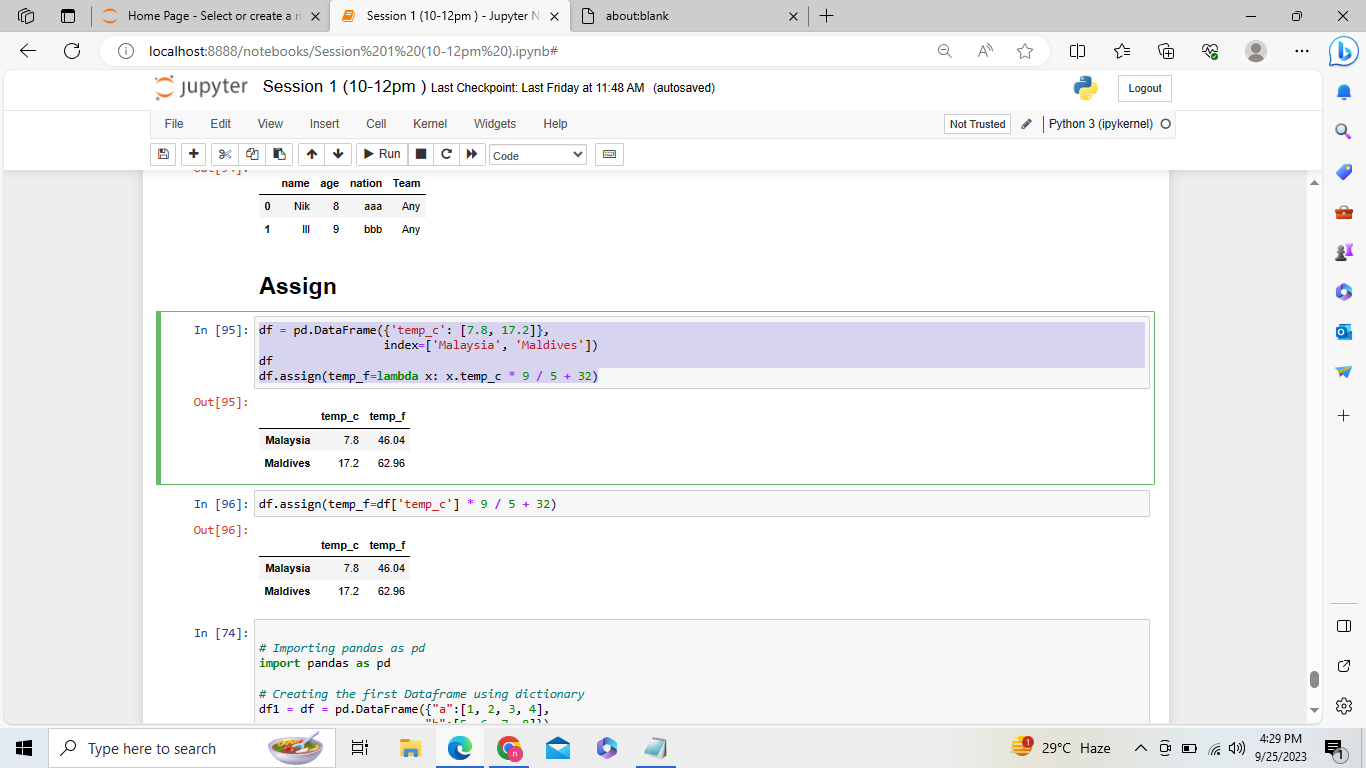
df = pd.DataFrame({'temp\_c': [99, 100]},

index=['Malaysia', 'Maldives'])

df

df.assign(temp\_f=lambda x: x.temp\_c \* 9 / 5 + 32)

Or df.assign(temp\_f=df['temp\_c'] \* 9 / 5 + 32)



## Read CSV Files

A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

In our examples we will be using a CSV file called 'data.csv'.

[Download data.csv](https://www.w3schools.com/python/pandas/data.csv). or [Open data.csv](https://www.w3schools.com/python/pandas/data.csv.txt)

### Example

Load the CSV into a DataFrame:

import pandas as pd

df = pd.read\_csv('data.csv')

#print(df.to\_string())

print(df.head())

print(df.tail())

print(df.info())

Df.size

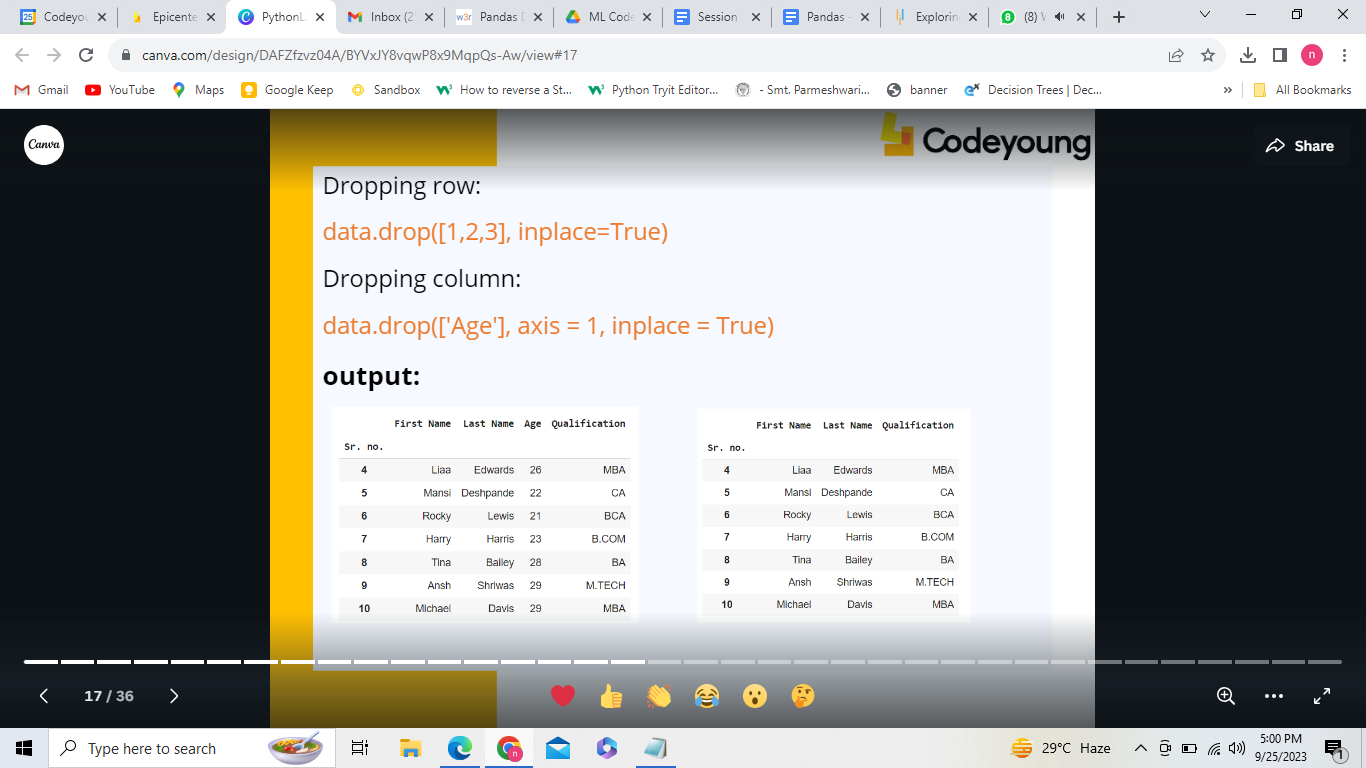
Df.columns

Df.shape

df.describe()

Sample dataset link:

<https://drive.google.com/file/d/1h30kN4Nn1Ro2FZth40GQYXbaCOmEQ0Sj/view?usp=share_link>



new1=pd.read\_csv("Sample\_dataset.csv")

#Drop rows

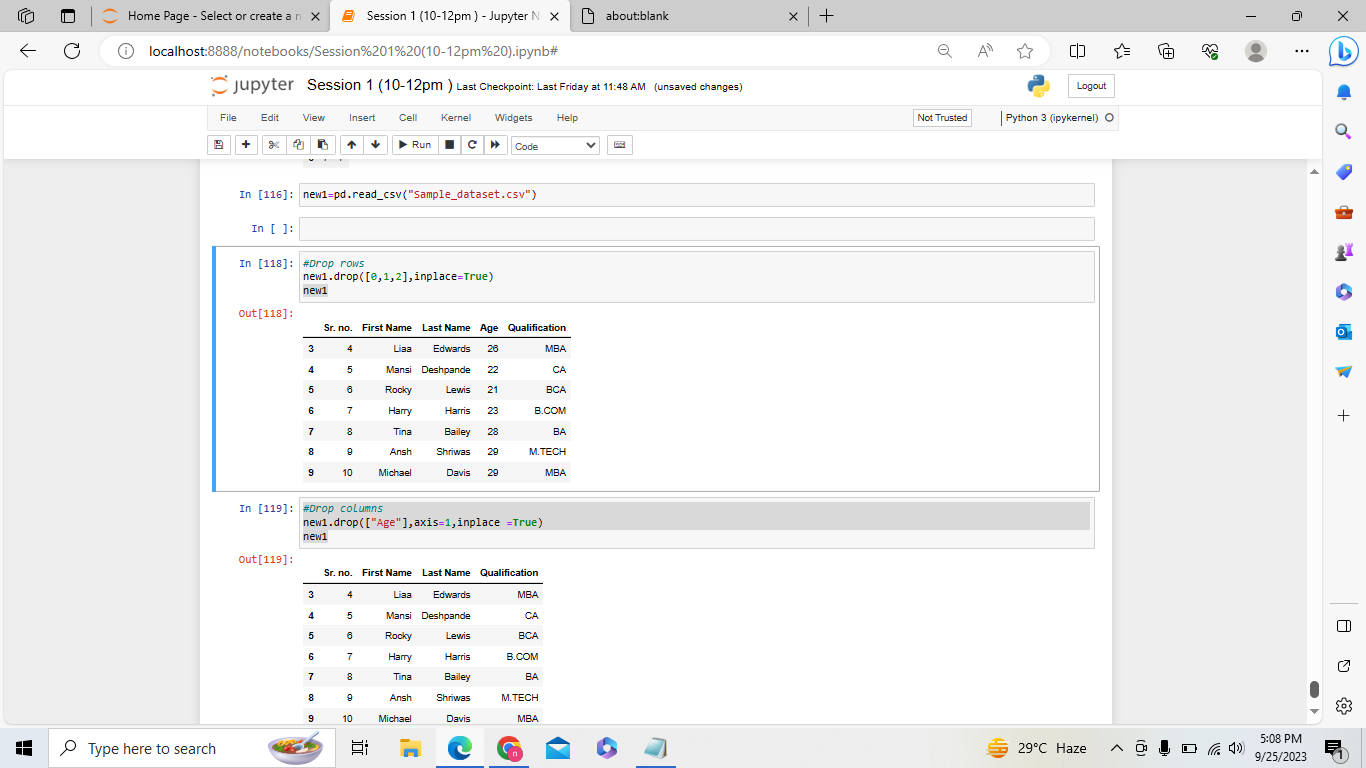
new1.drop([0,1,2],inplace=True)

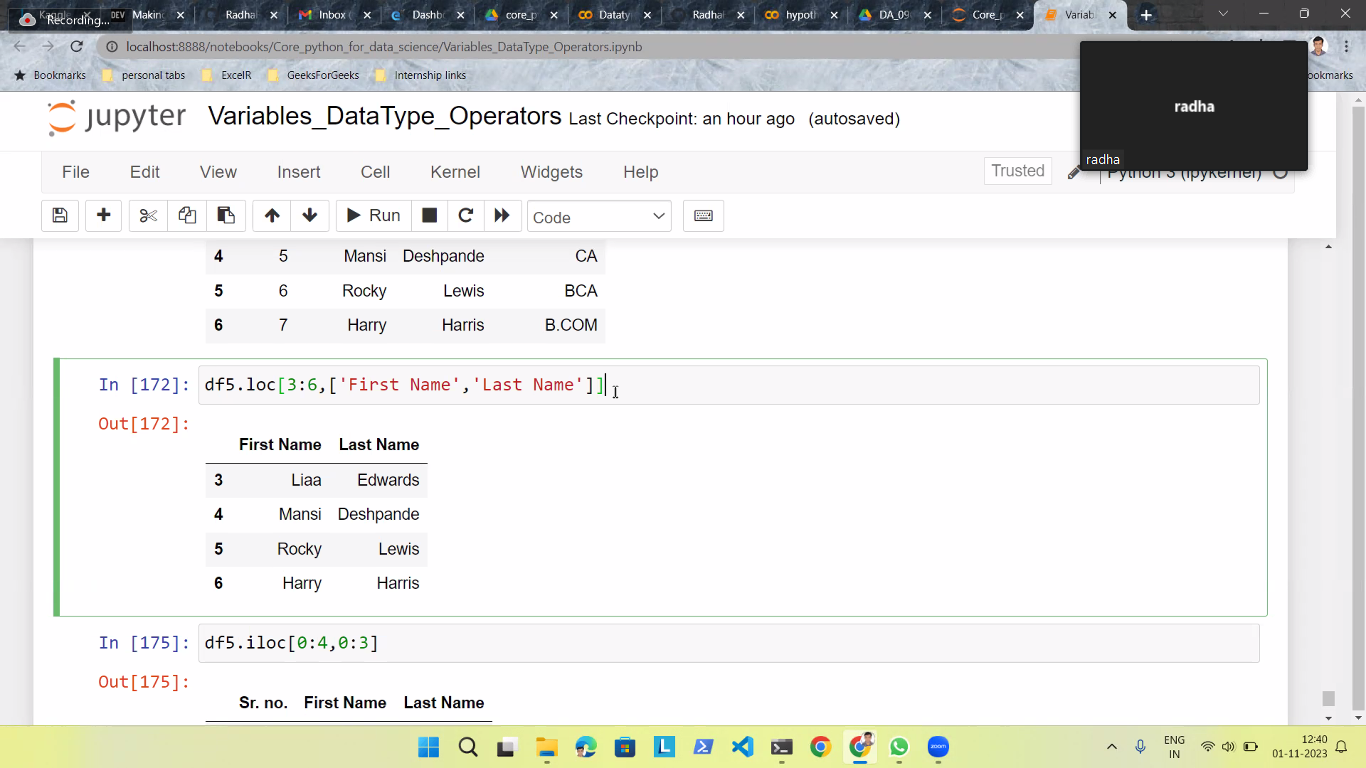
new1

#Drop columns

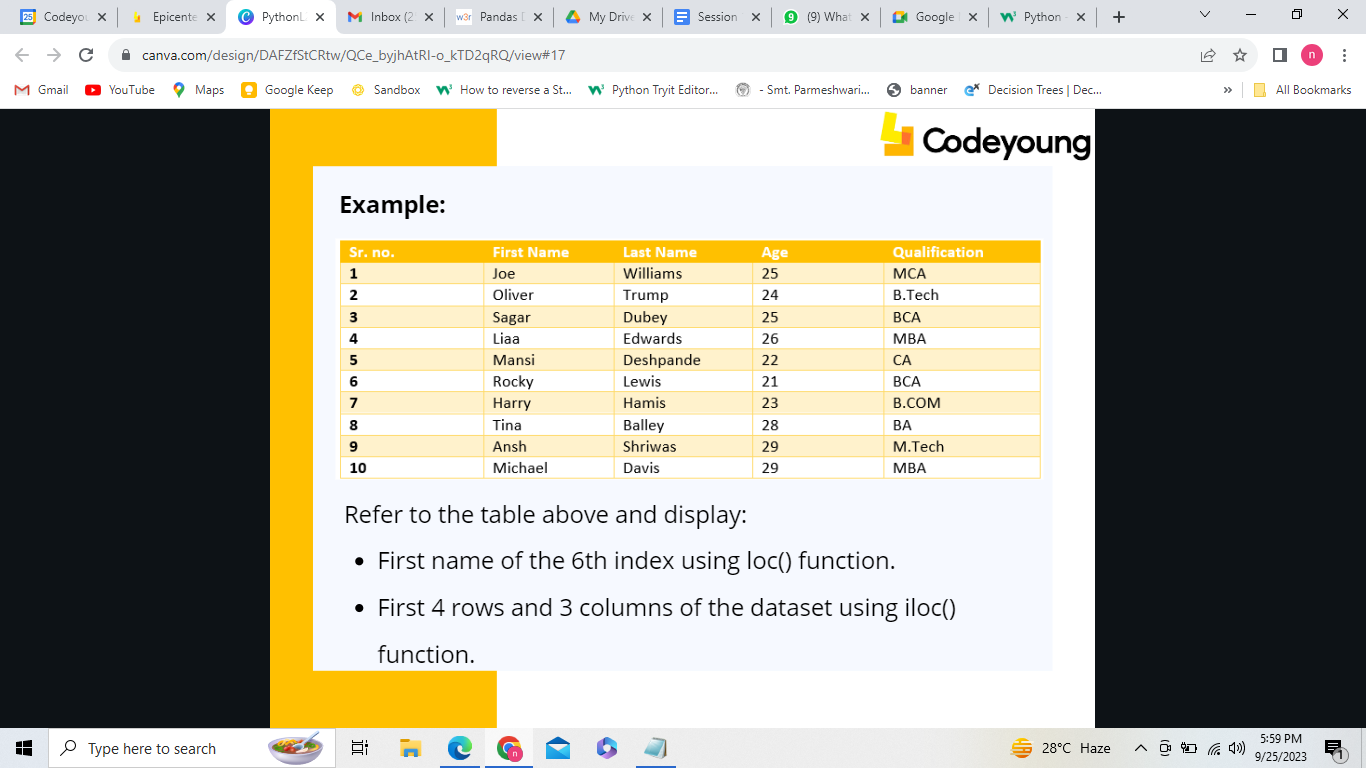
new1.drop(["Age"],axis=1,inplace =True)

New1





* Indexing in pandas is selecting particular rows and  
  columns of data from a dataframe.
* The **loc( ) function** is label based data selecting method  
  in which the label of selected columns is passed.
* The **iloc() function** is an indexed-based selecting  
  method, in which an integer index is passed to select a  
  specific row/column.



**Example:**

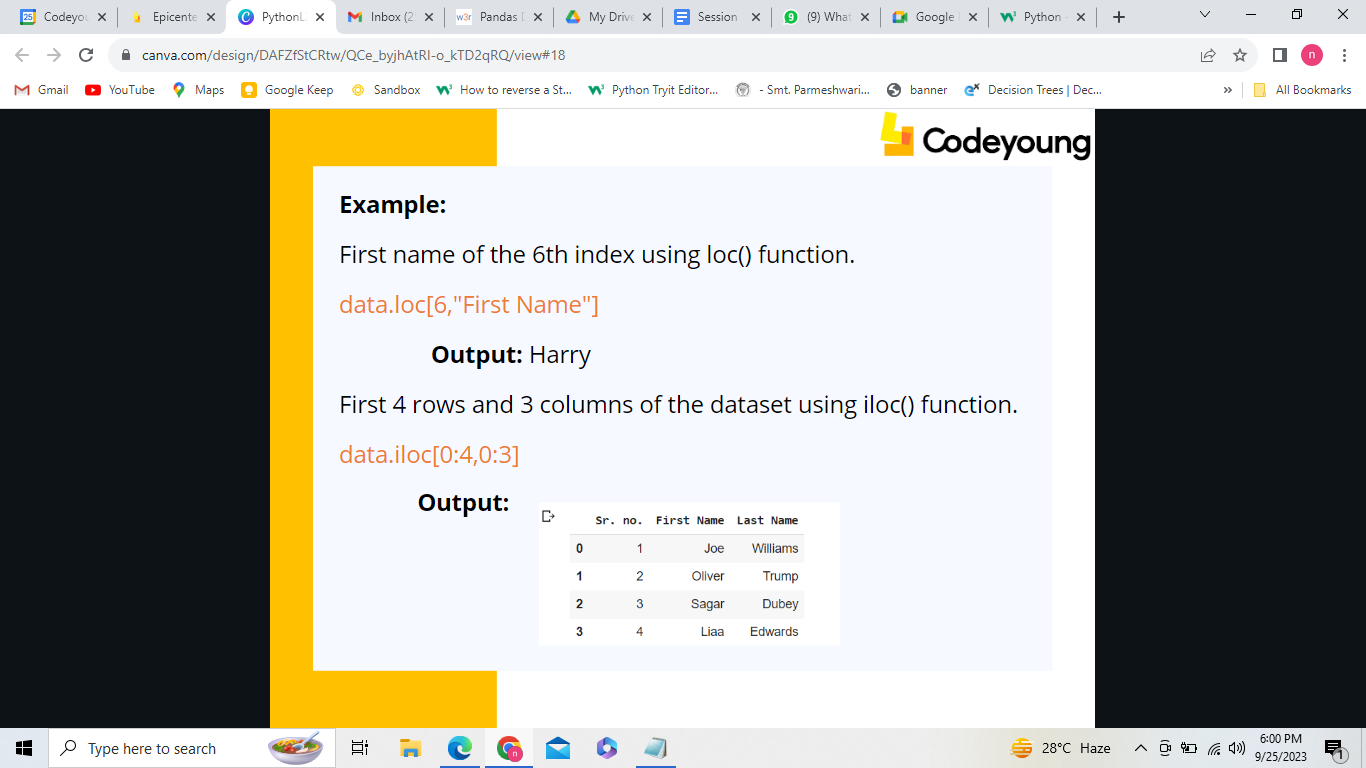
First name of the 6th index using loc() function.

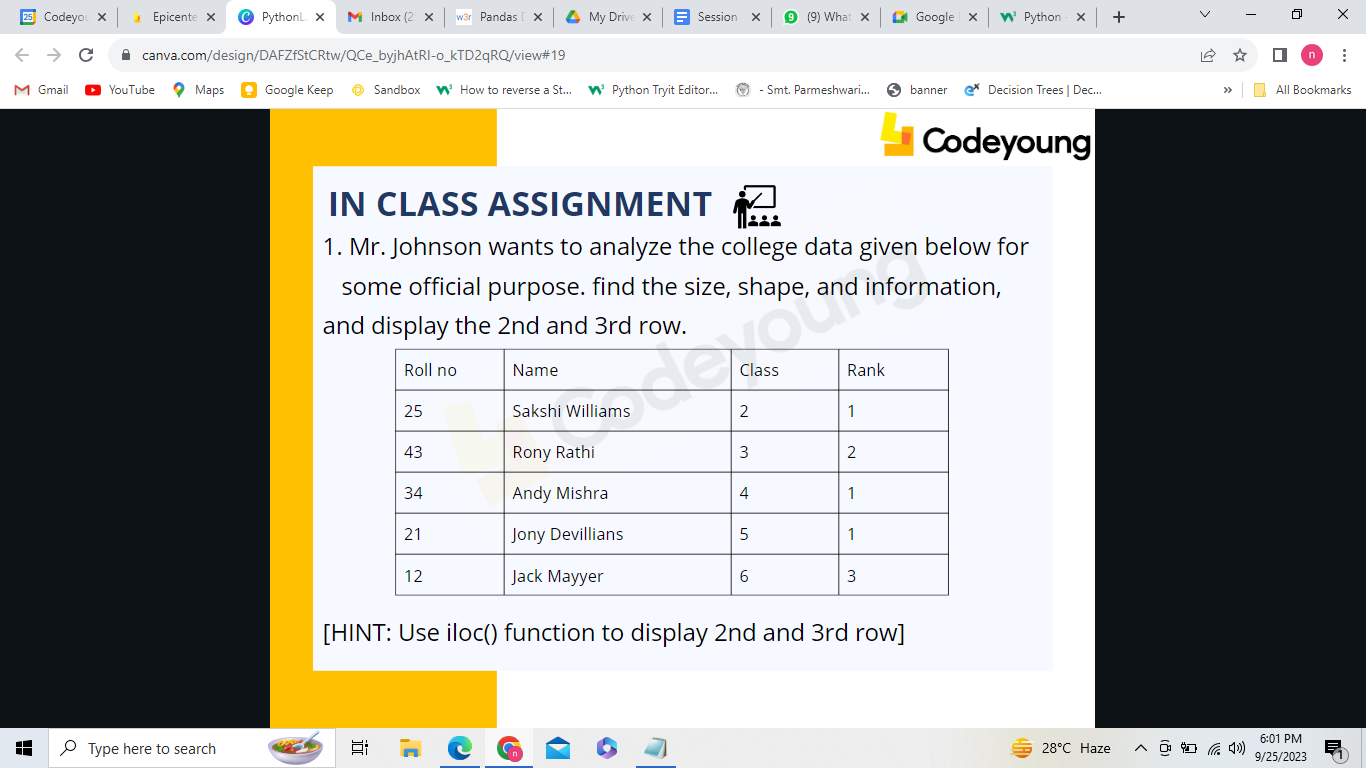
data.loc[6,"First Name"]

**Output:** Harry

First 4 rows and 3 columns of the dataset using iloc() function.

data.iloc[0:4,0:3]





########################################################################

## What is NumPy?

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra,and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

## Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

## Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

This behavior is called locality of reference in computer science.

This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

## Which Language is NumPy written in?

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)

## NumPy as np

NumPy is usually imported under the np alias.

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

## Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.

### Example

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))

## 0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

### Example

Create a 0-D array with value 42

import numpy as np

arr = np.array(42)

print(arr)

## 1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

### Example

Create a 1-D array containing the values 1,2,3,4,5:

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr

## 2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

NumPy has a whole sub module dedicated towards matrix operations called numpy.mat

### Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)

## 3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

### Example

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)

## Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

### Example

Get the first element from the following array:

import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])

### Example

Get third and fourth elements from the following array and add them.

import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[2] + arr[3])

## Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the row represents the dimension and the index represents the column.

### Example

Access the element on the first row, second column:

import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])

<https://www.w3schools.com/python/numpy/numpy_array_indexing.asp>

<https://towardsdatascience.com/pandas-most-used-functions-in-data-science-51b7c2b9c38a>

Array join:<https://www.w3schools.com/python/numpy/numpy_array_join.asp>

Matrix function:

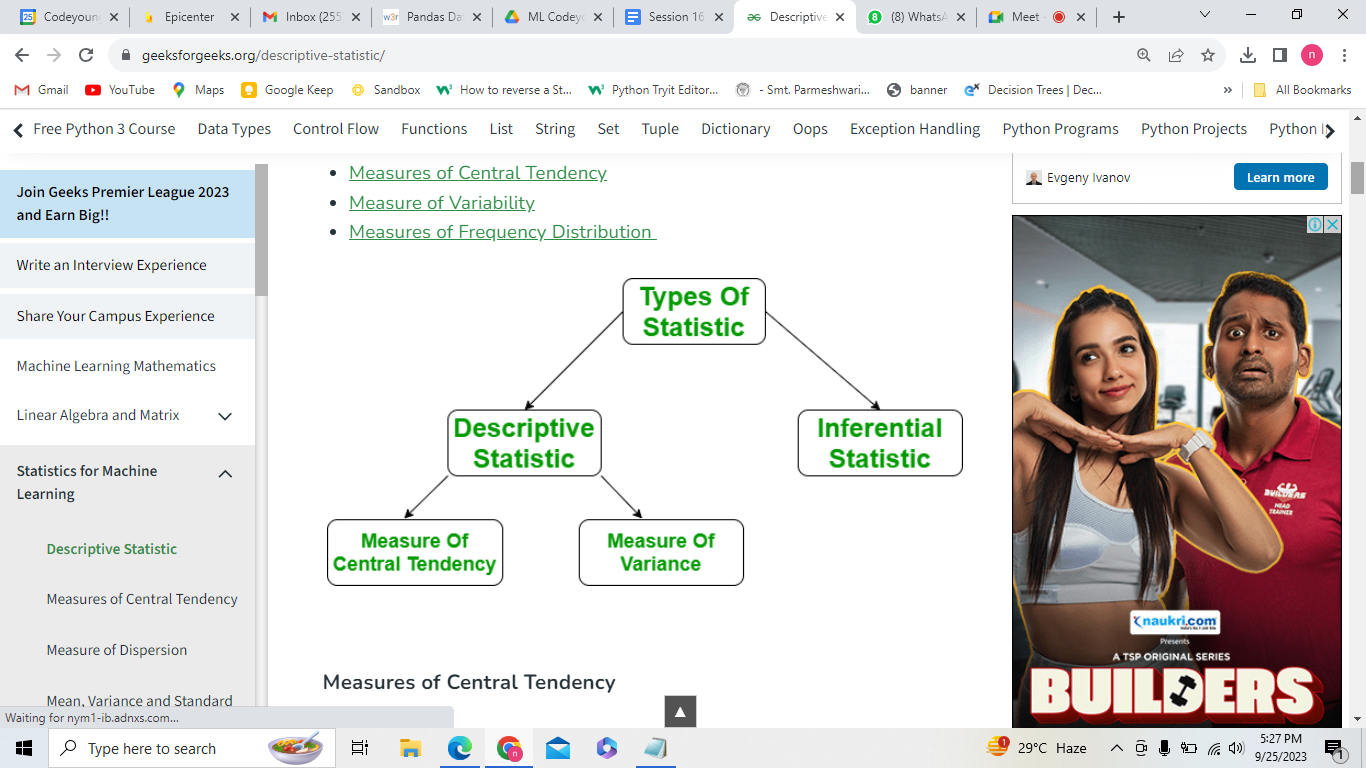
<https://www.geeksforgeeks.org/matrix-manipulation-python/>

## **What are Descriptive Statistics?**

In Descriptive statistics, we are describing our data with the help of various representative methods using charts, graphs, tables, excel files, etc. In descriptive statistics, we describe our data in some manner and present it in a meaningful way so that it can be easily understood. Most of the time it is performed on small data sets and this analysis helps us a lot to predict some future trends based on the current findings. Some measures that are used to describe a data set are measures of central tendency and measures of variability or dispersion.

## **Types of Descriptive Statistics**

* [Measures of Central Tendency](https://www.geeksforgeeks.org/measures-of-central-tendency/)
* [Measure of Variability](https://www.geeksforgeeks.org/measures-of-spread-range-variance-and-standard-deviation/)
* [Measures of Frequency Distribution](https://www.geeksforgeeks.org/frequency-distributions/)



**import** numpy as np

# Sample Data

arr **=** [5, 6, 11]

# Mean

mean **=** np.mean(arr)

print("Mean = ", mean)

Mode:# Mode

Import statistics as stats

mode **=** stats.mode(arr)

print("Mode = ", mode)#[1,1,2,3,1,4,1,6]

Median:

median **=** np.median(arr)# 1,2,3,8,4,5

print("Median = ", median)

#### **Range**

The range describes the difference between the largest and smallest data point in our data set. The bigger the range, the more the spread of data and vice versa.

*Range = Largest data value – smallest data value*

**import** numpy as np

# Sample Data

arr **=** [1, 2, 3, 4, 5]

# Finding Max

Maximum **=** max(arr)

# Finding Min

Minimum **=** min(arr)

# Difference Of Max and Min

Range **=** Maximum**-**Minimum

print("Maximum = {}, Minimum = {} and Range = {}".format(

Maximum, Minimum, Range))

#### **Variance**

It is defined as an average squared deviation from the mean. It is calculated by finding the difference between every data point and the average which is also known as the mean, squaring them, adding all of them, and then dividing by the number of data points present in our data set.

\sigma ^ 2 = \frac{\sum\left(x-\mu \right )^2}{N}

where,

* **x** -> Observation under consideration
* **N** -> number of terms
* **mu** -> Mean
* **import** statistics
* # sample data
* arr **=** [1, 2, 3, 4, 5]
* # variance
* print("Var = ", (statistics.variance(arr)))

**Output:**

Var = 2.5

Or:

a=np.var(ar)

a

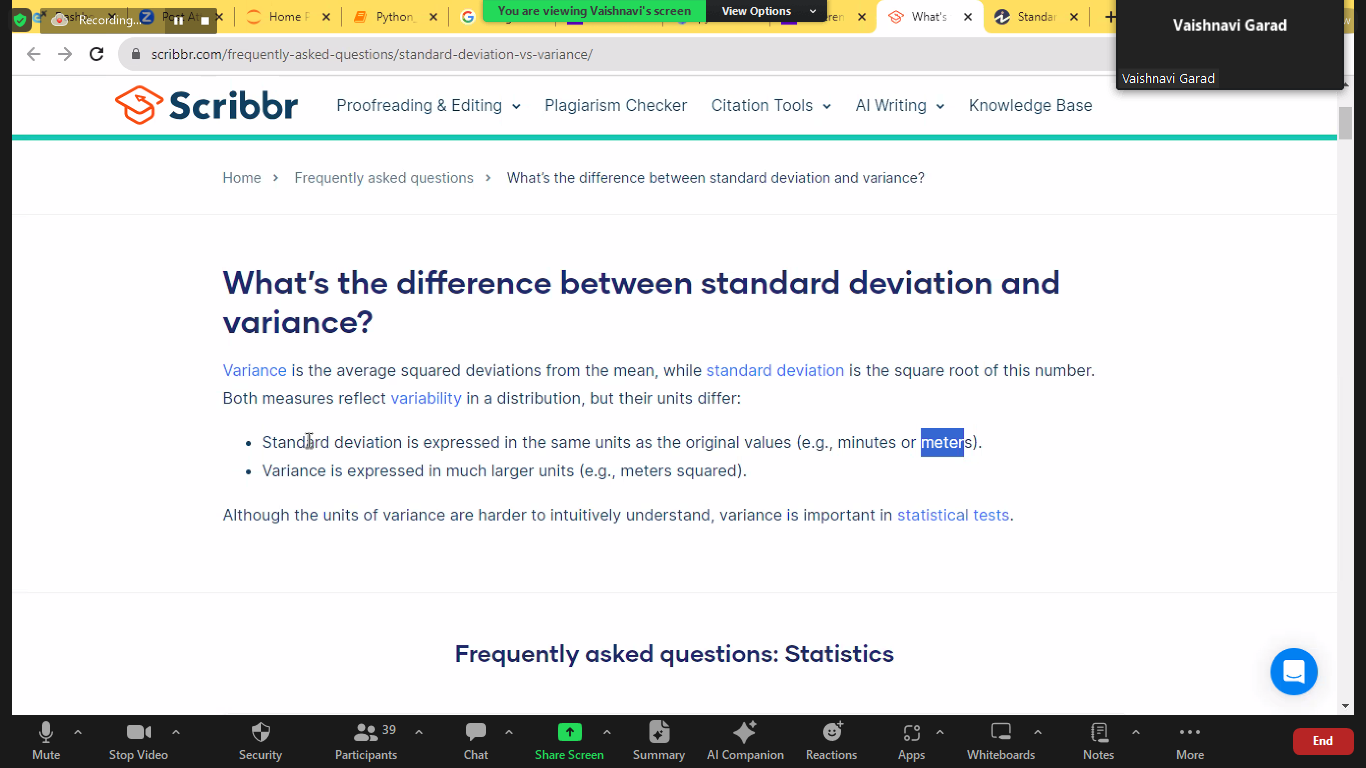
#### **Standard Deviation**

It is defined as the square root of the variance. It is calculated by finding the Mean, then subtracting each number from the Mean which is also known as the average, and squaring the result. Adding all the values and then dividing by the no of terms followed by the square root.

\sigma = \sqrt{\frac{\sum \left(x-\mu \right )^2}{N}}

where,

* x = Observation under consideration
* N = number of terms
* mu = Mean
* **import** statistics
* # sample data
* arr **=** [1, 2, 3, 4, 5]
* #st=np.std(arr)
* # Standard Deviation
* print("Std = ", (statistics.stdev(arr)))
* Out[31]:
* 1.6613247725836149



**Output:**

Std = 1.5811388300841898

* **1. PANDAS Stands for ?**
* a) Python dashboard analysis
* b) Python data scale
* c) Python data analysis
* d) Python dashboard system
* 2. **Which of the following statement will import pandas?**
* a)import pandas as pd
* b)import pandas as py1
* c)import pandas as ps
* d)all of the above
* **3. Which of the following is the advantage of  
  using pandas?**
* a) Data manipulation and analysis are quick  
  and efficient.
* b)Loading data from various file objects is  
  possible.
* c)Merging and connecting the datasets.
* d)All of these

**4. Find the type of error in the code below:**

* **import pandas as pd**
* **s =pd.Series([1,2,3,4,5],index= ['a','b','c','d','e'])**
* **print(s['f'])**
* a)Value error
* b)syntax error
* c)Index error
* d)No error

<https://www.w3schools.com/python/python_ml_standard_deviation.asp>