**PYTHON for ML**

**Start Date: May 1st 2020**

* **Installation:**
  + Download Python from its official website.
  + Download plugin Python by Microsoft in VS Code.
* **Key Points for arithmetic:**
  + 22/7 = 3.14……
  + 22//7 = 3 (floor division. Discards numbers after decimal)
* **There are no characters. Only Strings are available**
  + To add “ ” in strings, write
    - print(‘Hello “World”’) Output: Hello “World”
* **Variables and Arithmetic:**
  + To check the type of variables, type(variablename)
  + To get exponential, var1\*\*var2
* To take input from users, var=input(). Input always returns a string.
* There are no {} for functions or loops in python.
* **String:**
  + .format function:
    - It is used to format a string.
      * msg=var1+’ ‘+var2+’. How are you?’ can be written as
        + msg='{} {}. How are you?'.format(var1,var2)
        + msg=f'{var1} {var2}. How are you?' (f function)
  + To lower case – msg.lower()
  + To upper case – msg.upper()
  + To get the length – msg.\_\_len\_\_() or can also use len(msg)
  + To count how many times a character repeats, msg.count(‘character’)
  + To check if the word exists, msg.find(‘word’). Returns index.
* **Functions:**
  + Syntax:
    - def function\_name():
* **Modules:** 
  + Files can be created as a modules so that it can be shared. (Packages in flutter).
  + If the name of module file is “modules.py”
  + It can be imported as
    - import modules
    - To call the functions, module\_name.function\_name()
  + It can also be imported as
    - from module\_name import \*
    - To call the function, function\_name()
* **Conditional Statements:**
  + **If-else**

name = input("Please enter your name: ")

age = int(input("{}, Can you also enter your age: ".format(name)))

if age < 18:

    print("\n{}, sorry!! You cannont vote.".format(name))

else:

    print("\nGreat!! {}. You can vote.".format(name))

* + **If-elif-else**

if age < 18:

    print("\n{}, sorry!! You cannont vote.".format(name))

elif age > 18 and age < 21:

    print("\nGreat!! {}. You can vote because your age is {}".format(name, age))

else:

    print("\nGreat!! {}. You can vote.".format(name))

* **break and continue:**
  + Same syntax as C++/Java
* **For Loop:**
  + Syntax:
    - for i in range(1,20):

print(“Hello”)

* + - Here, 1 is the starting index and 20 is the ending index.
    - If we don’t provide starting index, by default is 0
    - If we want to provide how much ‘i’ should be incremented after each iterations, we can provide as
      * for i in range (1,20,3) where 3 is the incrementation
      * If we use this, we will have to compulsory use the starting index.
  + If we print the value of i as:
    - print(‘Value of i is “+i), it will give error. Concatenation is possible only for string. So, print(‘Value of i is “+str(i))
  + **To print pattern:**

for i in range(0, 10):

    for j in range(0, i):

        print('\*', end=" ")

    print('\r')

**\r moves to the next line and \n leaves a line**

**end is used to remove the next line character and end the output with a custom character.**

* **To check if a char/number is in a particular range,** 
  + If number in ‘0123456789’
* **while loop**
  + Syntax:

a = int(input("Enter a number: "))

b = 0

while b < a:

    print(b)

    b += 1

**there’s no ++ or -- operator in Python**

* **Lists**
  + A type of array but can store different type of data types.
  + Syntax:
    - list\_name = [1,’2’,’a’,3.14,True,None]
  + Slicing is the process of displaying or accessing a sublist from a list,
    - list\_name[starting\_index : ending\_index]
  + To access the list from the last index, you can use -ve. To access last element, list\_name[-1]
  + If two lists have same data but different order, both lists are counted as different
  + Multi-Dimensional List:

list\_name = [[1, 2, 3], ['a', "Abhishek"], [3.14], [True, None]]

for i in list\_name:

    print(i)

* + **Inserting into list**

spam = ['cat', 'rat', 'bat']

spam.append('mat')

spam.insert(0, 'mat')

print(spam)

append adds the item at the end of the list and insert adds the item at the specified position.

* + **Removing from list –** spam.remove(‘cat’)
  + **Sorting the list –** spam.sort()
* **Iterators:**
  + Iterators are known as data streams from which data can be extracted one by one.

string = '0123456789'

for i in iter(string):

    print(i)

days = ['Mon', 'Tue', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun']

days\_iter = iter(days)

for char in range(0, len(days)):

    next\_day = next(days\_iter)

    print(next\_day)

* **Dictionaries:**
  + To store different types of data, list is not normally used.
  + Dictionary is used to store different types of data.
  + Values in dictionary is stored in key-value pair
  + Two dictionaries with same values but different orders are counted as equal.
  + Syntax:

myCat = {'size': 'fat', 'color': 'black'}

print(myCat['size'])

* + If you want to print only values,

details = {'First Name': 'Abhishek', 'Surname': 'Doshi'}

for i in details.values():

    print(i)

* + If you want to print only key values, use details.keys()
  + To display both, use details.items()
  + Example:

details = {'Abhishek': '26/01/1999', 'Kush': '07/05/2003'}

while True:

    name = input("Enter Name: ")

    if name in details:

        print('Birthdate: '+details[name])

        break

    else:

        print('Name not in the list!!\n')

        bday = input('Enter the birthdate: ')

        details[name] = bday

        print("List Updated\n\n")

**Numpy**

* **Installation:**
  + Go to C:\Users\adosh\AppData\Local\Programs\Python\Python38\Scripts.
  + Open CMD
  + Write pip install numpy
* **Numpy is basically used for mathematical calculations especially related to arrays and matrices. Numerical Python Library.**
* **np.arange(starting\_index,ending\_index)**
  + This gives an array that contains numbers starting from starting\_index and ending to ending\_index
  + To get even number np.arange(0,10,2) where 2 is the number of incrementation.
* To generate 0 matrix, np.zeros(index) where index is the dimension of array.
* To generate m\*n matrix, np.zeros((m,n))
* To generate 1 matrix, np.ones(index)
* To generate random numbers:
  + np.random.randint(start,end)
  + To generate array of random numbers:
    - np.random.randint(start,end,(m\*n))
  + To generate random numbers without numpy:
    - x = random.sample(range(0, 10), 10)
  + To generate constant random numbers:

np.random.seed(1)

print(np.random.randint(0, 100, 10))

where 10 is the number of random numbers required.

* To get maximum of the array, array\_name.max()
* To get minimum, array\_name.min()
* To get average from array, array\_name.mean()
* To get the index of max value, array\_name.argmax()
* To get the index of min value, array\_name.argmin()
* To reshape an array into a matrix, eg from 1\*1 to 2\*5, use array\_name.reshape(m\*n)
* For getting a specific range of matrix, array\_name[start1:end1,start2:end2]

**MatPlotLib**

* Matplotlib is a library that is used to plot graphs in python.
* **Installation:**
  + Go to C:\Users\adosh\AppData\Local\Programs\Python\Python38\Scripts
  + Open cmd here.
  + Write pip install matplotlib
* **Importing matplotlib:**
  + import matplotlib.pyplot as plt
  + where matplotlib is the module and pyplot is the class inside matplotlib
* **Example:**

import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5, 6]

y = [2, 9, 8, 10, 20, 30]

plt.plot(x, y)

plt.xlabel('x axis')

plt.ylabel('y axis')

plt.title('First Plot')

plt.show()

* + Here, **plt.plot()** takes up the data for x and y axis
  + **plt.xlabel()** sets the label text for x axis
  + **plt.ylabel()** sets the label text for y axis
  + **plt.title** sets the title of the plot
  + **plt.show()** makes the graph visible.
* **Line Plot:**

import numpy as np

import matplotlib.pyplot as plt

time = np.arange(0.0, 2.0, 0.01)

s = 1 + np.cos(2\*np.pi\*time)

plt.plot(time, s,)

plt.grid()

plt.xlabel('Time')

plt.ylabel('Voltage (mV)')

plt.title('Cosine Wave Plot (cos(x))')

plt.show()

* + **plt.grid()** provides grid in the background
  + 
* **Subplot**
  + Making multiple plots in same plane

import numpy as np

import matplotlib.pyplot as plt

x1 = np.linspace(0.0, 5.0)

x2 = np.linspace(0.0, 2.0)

y1 = np.cos(2\*np.pi\*x1)\*np.exp(-x1)

y2 = np.cos(2\*np.pi\*x2)

plt.subplot(2, 1, 1)

plt.plot(x1, y1, 'o-')

plt.title('Subplot 1')

plt.xlabel('x1')

plt.ylabel('y1')

plt.subplot(2, 1, 2)

plt.plot(x2, y2, '--')

plt.title('Subplot 2')

plt.xlabel('x2')

plt.ylabel('y2')

plt.show()

* + **np.linspace()** provides a linear range starting from 0
  + **plt.subplot(row,column,index)** defines where to plot the subplots



* **Bar Plot:**
  + When we want to measure data only on y-axis, we use bar plot.

import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]

y = [10, 24, 36, 40, 5]

label = ['One', 'Two', 'Three', 'Four', 'Five']

plt.bar(x, y, tick\_label=label, width=0.8, color=['green', 'blue', 'black'])

plt.xlabel('X Axis')

plt.ylabel('Y Axis')

plt.title('Bar Graph')

plt.show()

* + tick\_label is used to give labels to each bar on x-axis
* **Histogram:**
  + It is used for distribution of frequency on one variable.

import matplotlib.pyplot as plt

age = [2, 50, 10, 70, 8, 21, 22, 30, 40, 64, 78, 20,

       36, 78, 0, 85, 20, 36, 90, 8, 75, 21, 56, 32, 14]

range = (0, 100)  # Range of data

bins = 10  # Range for Y-axis as histogram works only on x-axis.

plt.hist(age, bins, range, color='green', histtype='bar', rwidth=0.7)

# histtype is type of histogram and rwidth is width between the bars

plt.xlabel('Ages')

plt.title('Histogram')

plt.ylabel('Bins')

plt.show()



* **Scatter Plot:**
  + It is a point graph
  + It is used when we want to find the data for a particular value

import matplotlib.pyplot as plt

import numpy as np

import random

x = random.sample(range(0, 10), 10)

y = random.sample(range(0, 10), 10)

# label is the one that gives detail about marker

plt.scatter(x, y, label='Data', color='green', marker='\*', s=50)

plt.title('Scatter Plot')

plt.xlabel('X Axis')

plt.ylabel('Y Axis')

plt.legend()  # legend is the box that displays marker and label

plt.show()



* **Pie Chart:**
  + It is used to display graph in circular form and displaying as % wise.

import matplotlib.pyplot as plt

activities = ['TOC', 'AJ', '.Net', 'WT', 'SE']

slices = [3, 7, 8, 6, 5]

plt.pie(slices, labels=activities, startangle=90,

        explode=(0.2, 0, 0, 0, 0), autopct='%1.2f%%')

# explode gives the slice out of the chart.

# autopct gives % automatically and %1.2%% means that 2 values after decimal

plt.legend()

plt.show()



* **Merging plots in single plane:**

import matplotlib.pyplot as plt

import numpy as np

x = np.arange(0, 2\*np.pi, 0.01)

y1 = np.sin(x)

y2 = np.cos(x)

plt.plot(x, y1, label='sin')

plt.plot(x, y2, label='cos')

plt.xlabel('X-Axis')

plt.ylabel('Y-Axis')

plt.title('Sin and Cos Functions')

plt.legend()

plt.show()



* **Plotting Data from CSV and txt file:**

import matplotlib.pyplot as plt

import csv

x = []

y1 = []

y2 = []

with open('D:\python-programs\Plot using csv\data.txt', 'r')as datafile:

    plots = csv.reader(datafile)

    for col in plots:

        x.append(col[0])

        y1.append(col[1])

with open('D:\python-programs\Plot using csv\data.csv', 'r')as datafile:

    plots = csv.reader(datafile)

    for col in plots:

        y2.append(col[1])

plt.plot(x, y1, label='Text File Data')

plt.plot(x, y2, label='CSV File Data')

plt.xlabel('X Axis')

plt.ylabel('Y Axis')

plt.title('Plot using txt and csv file')

plt.legend()

plt.show()



**Pandas**

* Data Analysis Library
* It is used to preprocess the data.
* We can preprocess the data that has half data, or unproper data or false data.
* Raw data to processed data.
* Algorithms can’t be applied if data is not processed.
* **Installation: pip install pandas**
* **Series Object:**
  + It is used to show a list or dictionary in more sequential way

import pandas as pd

subjects = ['TOC', 'AJ', 'WT', 'SE', 'Dot Net']

pd.Series(subjects)

print('\nData without series object: ', subjects)

print('\nData with series object: \n', pd.Series(subjects))

Output:



* **Attributes:**
  + They are used to extract data from Series.
  + Attributes are just used to extract and not to operate on them.

import pandas as pd

men = ['Smart', 'Intelligent', 'Caring', 'Loving']

s = pd.Series(men)

print(s.values)  # Displays values as array

print(s.index)  # Displays start and end index

print(s.dtype)  # Displays data type

* **Methods:**

import pandas as pd

rate = [10, 20, 3.95, 7.85, 4.65, 1.26, 3.84, 2.79]

index = [10, 20, 30, 40, 50, 60, 70, 80]

r = pd.Series(rate)

print('Sum of all values: ', r.sum())

print('Product of all values: ', r.product())

print('Average of all values: ', r.mean())

print(pd.Series(rate, index, name='List'))

* + Output:



* **Read CSV Files using Pandas:**

import pandas as pd

x = pd.read\_csv('data.csv', squeeze=True, usecols=['X-Axis'])

# squeeze converts csv file into panda series.

# usecols defines which column yoou want to access

print(x.head())  # Extracts  values from start

print(x.head(7))  # Extracts 7 values from start

print(x.tail())  # Extracts 5 values from end

print(x.values)  # Displays only values

#Checks if there are any repeated values or not. Returns True if all are unique

print(x.is\_unique)

# To get the dimension of series

print(x.ndim)

# To print number of rows and columns

print(x.shape)

print(x.size)

# Prints column name

print(x.name)

* **Sorting Values of CSV:**

import pandas as pd

y = pd.read\_csv('data.csv', squeeze=True, usecols=['Y-Axis'])

print(y.sort\_values())  # Ascending

print(y.sort\_values(ascending=False))  # Descending

* **Inplace attribute:**

import pandas as pd

y = pd.read\_csv('data.csv', squeeze=True, usecols=['Y-Axis'])

y.sort\_values(inplace=True, ascending=False)

# inplace replaces the original list with the sorted list.

# It is like: y=y.sort(ascending=False)

print(y)

* **Sort\_index:**
  + It helps to resort the series if you have messed up with the series

import pandas as pd

y = pd.read\_csv('data.csv', squeeze=True, usecols=['Y-Axis'])

y.sort\_values(ascending=False, inplace=True)

# Now the value of y is the series in descending order

y.sort\_index(inplace=True)

# Now the data is again back to the old one.

print(y)

* **Extracting data using index position:**

import pandas as pd

y = pd.read\_csv('data.csv', squeeze=True, usecols=['Y-Axis'])

# Extract data using index

test = y[5]

# Extracting multiple values

test = y[[5, 7, 8]]

# Extracting series

test = y[5:10]

# Extracting last 6 values

test = y[-6:]

* **Extracting data using index label:**

import pandas as pd

y = pd.read\_csv('data.csv', squeeze=True, index\_col='X-Axis')

test = y["One"]

print(test)

* **get method:**

import pandas as pd

y = pd.read\_csv('data.csv', squeeze=True, index\_col='X-Axis')

y.sort\_index(inplace=True)

# Displays data of One and Three

test = y.get(["One", "Three"])

print(test)

# Displays the default value if there's no data.

test = y.get('Zero', default='Data Not Available!!')

print(test)

* **Math Methods in Pandas:**

import pandas as pd

y = pd.read\_csv('data.csv', squeeze=True, usecols=['Y-Axis'])

print(y.count())  # Counts number of rows

print(len(y))  # Counts number of rows

print(y.sum())  # Adds all data

print(y.mean())  # Calculates Average

print(y.std())  # Calculates Standard Deviation

print(y.min())  # Prints minimum value

print(y.max())  # Prints maximum value

print(y.median())  # Prints Median

* **Idxmax() and idxmin():**
  + It is used to display the index of minimum and maximum value of Series.

import pandas as pd

y = pd.read\_csv('data.csv', squeeze=True, usecols=['Y-Axis'])

print(y.idxmax())  # Displays index of maximum value

print(y.idxmin())  # Displays index of minimum value

* **value\_count():**
  + It count no. of existence.

import pandas as pd

y = pd.read\_csv('data.csv', squeeze=True, index\_col='X-Axis')

print(y.value\_counts())

* + It will give count of how many values are there for each category.
* **apply():**
  + To apply a particular function on a particular data

import pandas as pd

google = pd.read\_csv(

    'D:\python-programs\Datasets\google\_stock\_price.csv', squeeze=True)

def classify\_performance(number):

    if number < 300:

        return "OK"

    elif 650 > number >= 300:

        return "Satisfactory"

    else:

        return "Incredible"

print(google.apply(classify\_performance))

* **Anonymous Function:**

# Using Anonymous Function

print(google.apply(lambda stock\_price: stock\_price+1))

* + Here, all the data will be available in stock\_price
* **DataFrames:**
  + Series is 1-D Array of any dataset. DataFrames is multidimensional array of any dataset.
  + Some of the same methods & attributes of series and dataframes:
    - head()
    - tail()
    - index
    - shape
    - dtypes
  + Some functions & attributes specifically for dataframes:
    - columns – returns column as array
    - axes – gives columns along with index
    - info() – precise info about dataset
    - get\_dtype\_count() – counts number of each datatypes. However, it’s deprecated since v0.25.0
* **Adding data from DataFrames:**

import pandas as pd

rev = pd.read\_csv('D:\python-programs\Datasets\\revenue.csv', index\_col="Date")

print(rev.sum())  # Adds data column-wise

print(rev.sum(axis=0))  # 0 is used for Column wise adding

print(rev.sum(axis=1))  # 1 is used for Row wise adding

* **Shared Method & Columns:**

import pandas as pd

nba = pd.read\_csv('Datasets//nba.csv')

print(nba.head(11))

# Extract Data Column wise

# Name is the column name. Drawback is that the column name must not have space

print(nba.Name.head())

print(nba["Name"].head())  # Using List. Can have space.

columns = ["Name", "Team"]

print(nba[columns].head())  # Accessing multiple columns

* **Add new column to DataFrame:**

import pandas as pd

nba = pd.read\_csv('Datasets\\nba.csv')

nba['Sports'] = 'BasketBall'  # Adds a col 'Sports' with value Basketball

# However, it doesn't affect data in csv file

nba.insert(2, 'Sports', 'Basketball')  # nba.insert(index,column\_name,value)

print(nba.head())

* **Broadcasting Operations:**
  + If we want to perform an actions on an individual series of columns, we use broadcasting operations.

import pandas as pd

nba = pd.read\_csv('Datasets//nba.csv')

# Addition

print(nba['Salary'].add(50000))

print(nba['Salary']+50000)

# Subtraction

print(nba['Salary'].sub(50000))

print(nba['Salary']-50000)

# Multiplication

print(nba['Salary'].mul(2))

print(nba['Salary']\*2)

# Replace with another column

nba["Weight in KG"] = nba["Weight"]\*0.453592

print(nba.head())

* **Drop Rows with Null Values:**

import pandas as pd

nba = pd.read\_csv('Datasets\\nba.csv')

# Removes all the rows that contain any NaN or NULL Value for a column

print(nba.dropna())

# Removes the rows that contains all NULL values.

nba.dropna(how="all", inplace=True)

print(nba)

# Removes null values from a specific column

print(nba.dropna(subset=['Salary']))

* **fillna() method:**
  + To replace NA or NULL value with a specific value.

import pandas as pd

nba = pd.read\_csv('Datasets\\nba.csv')

# Replace all NULL values

nba.fillna(value=0, inplace=True)

# Replace all NULL Values in Salary

nba['Salary'].fillna(value=0, inplace=True)

# Replace all NULL Values in College

nba['College'].fillna(value="No College", inplace=True)

print(nba.head())

* **sort\_values() in DataFrame:**
  + NAN or NULL values go at last.
  + To bring NAN at top, there’s a parameter named na\_position in sort\_values().
  + For last, na\_position=’last’ (default) and for first, na\_position=’first’.

import pandas as pd

nba = pd.read\_csv('Datasets//nba.csv')

nba.sort\_values("Salary", ascending=False, inplace=True)

print(nba.head())

* **sort\_values() for multiple columns:**

import pandas as pd

nba = pd.read\_csv('Datasets//nba.csv')

# Sort according to team and inside team, name will also be sorted

nba.sort\_values(["Team", "Name"], inplace=True)

# Sort according to team and inside team, name will also be sorted descending

nba.sort\_values(["Team", "Name"], ascending=[True, False], inplace=True)

print(nba.head())

* + So, in this, it will first sort team and then name. So, there will be sorted according to team and inside team, name will also be sorted.
* **rank() method:**
  + It gives ranking according to the column specified.

import pandas as pd

nba = pd.read\_csv('Datasets\\nba.csv')

nba.dropna(how='all', inplace=True)

nba['Salary'].fillna(0, inplace=True)

nba['SalaryRank'] = nba['Salary'].rank(ascending=False).astype(int)

nba.sort\_values(by="Salary", ascending=False, inplace=True)

print(nba)

* + astype() converts data into specified datatype.
* **Filtering DataFrames:**
  + Main work of Filtering is, it reduces memory.
  + Filtering means, changing the existing data types of columns, to the correct data types

import pandas as pd

employees = pd.read\_csv('Datasets\employees.csv')

# Displays the info such as column data type.

print(employees.info())

# Now, the task is to change the data type of all the columns to correct datatype

# Change Start Date to date-time data format

employees['Start Date'] = pd.to\_datetime(employees['Start Date'])

# Change Login Time into date-time data format

employees['Last Login Time'] = pd.to\_datetime(employees['Last Login Time'])

# Change Senior Management into Boolean data format

employees['Senior Management'] = employees['Senior Management'].astype(bool)

# Change Gender into Category data format

employees['Gender'] = employees['Gender'].astype("category")

* + The above things can easily be done as:

import pandas as pd

employees = pd.read\_csv('Datasets\employees.csv', parse\_dates=[

                        "Start Date", "Last Login Time"])

employees['Senior Management'] = employees['Senior Management'].astype(bool)

employees['Gender'] = employees['Gender'].astype("category")

print(employees.info())

* + parse\_dates converts columns into date-time format
* **Filtering based on conditions:**

# Extract data that has Gender = Male

print(employees[employees['Gender'] == 'Male'])

# It can also be done as:

data = employees['Gender'] == 'Male'

print(employees[data])

* **Filtering with AND & OR (Operations on multiple columns):**
  + In python, AND is & and OR is |

data1 = employees['Gender'] == 'Male'

data2 = employees['Team'] == 'Marketing'

data3 = employees['Bonus %'] < 1.5

print(employees[data1 & data2 & data3])

* + So this will print the details of employee who is **Male** and is working in **Marketing Team** with **Bonus % less than 1.5**
* **isin() method:**

#Displays employees from Legal or Sales or Product team

print(employees[employees['Team'].isin(['Legal', 'Sales', 'Product'])])

* + isin() method is used to apply or condition in a single task.
  + It displays the data from a particular column which is in a particular data list.
* **isnull() and notnull() method:**
  + To get the data which contains NULL values, use isnull()
  + To get the data other than NULL values, use notnull()

print(employees[employees['Team'].isnull()])

* + - This prints all the data that has Team value NaN or NULL

print(employees[employees['Team'].notnull()])

* + - This prints all the data that has Team value not NULL
* **between() method:**
  + It is used to get values which is in between 2 values.

print(employees[employees['Bonus %'].between(15, 20)])

* + It displays data that has Bonus % between 15-20
* **duplicated() method:**
  + It prints duplicate values.
  + The 1st value will be considered as the new value. And after that, if there’s same values, it considers it as duplicate.
  + If we want to discard first also, we can specify it’s parameter, **keep=false**
  + To get last value as duplicate only, **keep=last**

print(employees[employees['First Name'].duplicated(keep=False)])

* + This will display all duplicates.
  + To display only unique values:

print(employees[~employees['First Name'].duplicated(keep=False)])

* **drop\_duplicates() method:**
  + It drops all the rows that have exactly same values.

print(employees.drop\_duplicates(subset=['First Name']))

* + - Drops all the rows which has duplicate First Name.

employees.drop\_duplicates(subset=['First Name', 'Team'], inplace=True)

print(employees)

* + - Drops all the rows which has duplicate First Name or Team
    - So it will display data that can have same First Name but has different Team
* **set\_index() and reset\_index():**
  + It is used to set and reset a specific column as index.

import pandas as pd

bond = pd.read\_csv('Datasets\jamesbond.csv')

bond.set\_index("Film", inplace=True)  # Does the same work of index\_col

print(bond)

bond.reset\_index()

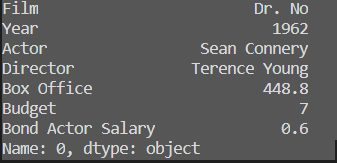
print(bond)

* **Retrieve row values using loc[]:**

data = bond.loc[0]

print(data)

* + Output:



* + It displays all data at 0th index.

data = bond.loc[0]['Film']

print(data)

#This can also be done as:

data=bond.loc[0,’Film’]

print(data)

* + This prints Film Name of 0th index.

data = bond.loc[[0, 1]]

print(data)

* + This prints data of 0th and 1st index.
* **Retrieve row values using iloc[]:**

print(bond.iloc[0:20:2])

* + The above statement prints data starting from 0 to 20 with a gap of 2 i.e. 0,2,4,6,8…20

bond = pd.read\_csv('Datasets\jamesbond.csv', index\_col='Film')

bond.sort\_index()

print(bond.iloc[0])

print(bond.loc['Dr. No'])

* + The above iloc and loc statements displays the details of Dr. No movie.
* **Set values for a specific cell:**

# Changes the name of Actor of Dr. No

bond.loc['Dr. No', 'Actor'] = 'Sir Sean Connery'

print(bond.loc['Dr. No'])

# Changes multiple values

bond.loc['Dr. No', ['Actor', 'Budget']] = ['Sean Connery', '10']

print(bond.loc['Dr. No'])

* **Set Multiple Values in DataFrame:**

bond.loc[bond['Actor'] == 'Sean Connery']['Actor'] = 'Sir Sean Connery'

* **Rename Index Values of Column:**

bond.rename(columns={'Year': 'Release Date',

                     'Box Office': 'Revenue'}, inplace=True)

* + columns property requires Dictionary. So, here, Year will be renamed to Release Date and Box Office is changed to Revenue

bond.rename(index={'Dr. No': 'Doctor No',

                   'GoldenEye': 'Golden Eye'}, inplace=True)

* + index property is used to rename a particular index. Dr. No will be renamed to Doctor No and GoldenEye will be renamed to Golden Eye.
  + The columns can also be renamed using:

bond.columns = ['Release Year', 'Actor',

                'Director', 'Revenue', 'Budget', 'Actor Salary']

* + - However, you cannot change the index column name in this.
* **Delete Rows or Columns using drop():**

bond.drop('A View to a Kill', inplace=True)

* + Deletes the row for ‘A View to a Kill’

bond.drop(columns={'Box Office'})

* + Deletes Box Office entire column. It can also be done as:

bond.drop(['Box Office', 'Budget'], axis=1)

* + To get the entire deleted column:

actor = bond.pop('Actor')

* + Column can also be deleted as:

del bond['Director']

* + - This doesn’t require inplace.
* **Create Random Samples:**

data = bond.sample()

* + Returns a single row randomly.

data = bond.sample(n=10)

* + Returns 10 rows randomly.

data = bond.sample(frac=0.25)

* + Returns 25% rows randomly.

data = bond.sample(n=2, axis=1)

* + Returns 2 columns randomly.
* **nsmallest() & nlargest():**
  + Extracts n number of smallest and largest values.

bond.nsmallest(n=3, columns='Box Office')

* + - Returns 3 rows which has least Box Office value.

bond.nlargest(n=3, columns='Box Office')

* + - Returns 3 rows which has highest Box Office value.

bond['Box Office'].nlargest(10)

* + - This will display only the Box Office data
* **where():**
  + It’s used to filter out data using a condition.
  + It returns all data, however fills NAN values where the condition doesn’t match.

bond.where(bond['Actor'] == 'Sean Connery')

* + This will give all data and the DataFrame will conatin data that has Actor as Sean Connery and the rest of the data will be NAN.
* **query():**
  + It works only when all the column name is string and doesn’t have space.

bond.columns = [column\_name.replace(' ', '\_') for column\_name in bond.columns]

print(bond.query(expr='Budget > 10.0'))

* + The 1st statement, replaces all the spaces in the column name with underscore.
  + The 2nd statement, returns all the rows with Budget > 10.0
* **apply():**
  + It is used to apply a condition entire to a series.

def add\_millions(number):

    return str(number)+' Millions'

bond['Box Office'] = bond['Box Office'].apply(add\_millions)

print(bond)

* + This adds Millions to the Box Office.
  + apply () can also be used on row:

def good\_movie(row):

    actor = row[1]

    budget = row[4]

    if actor == 'Sean Connery':

        return 'The best'

    elif actor == 'Roger Moore' and budget > 40:

        return 'Enjoyable'

    else:

        return 'Get Lost!'

bond.apply(good\_movie, axis='columns')

* + This will check the actor and budget of a particular movie and return the data accordingly.