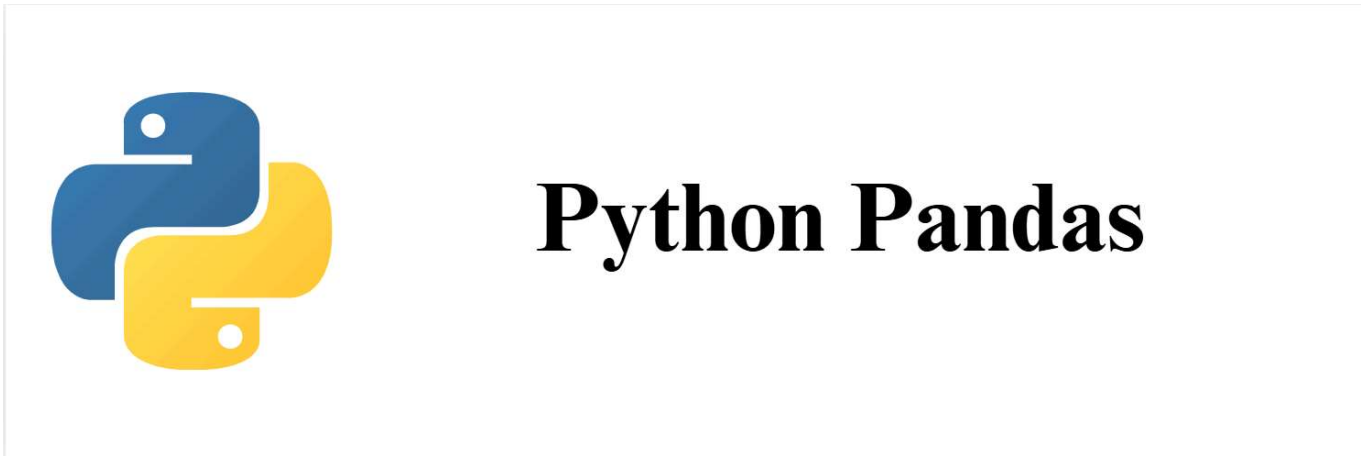


Pandas

Python library used for working with datasets. It is a python package that provides fast, flexible, and expressive data structures designed to make working with relational/labeled data both easy and initiative.

The most powerful and flexible open source analyzing, cleaning, exploring and manipulating data.

- Easy handling of missing data
- Size mutability(columns can be inserted and deleted)

```
import pandas as pd
pd.__version__
```

DataFrame

- 2D array or a table with rows and column

```
mydataset = {
  'Bike Category': ['Adventure','Street Bikes','Touring','Sports'],
  'Bike Names': ['GSA 1250','Speed_Twin','RE_350','S1000RR' ]
}

myvar=pd.DataFrame(mydataset)

print(myvar)
```

```
   Bike Category  Bike Names
0     Adventure    GSA 1250
1  Street Bikes  Speed_Twin
2       Touring      RE_350
3        Sports     S1000RR
```

iloc (integer-location based indexing)

Access data by specifying the rows and column positions as integers.

loc(Label-based indexing)

Access data by specifying the row and column labels.

```
#iloc
print(myvar.iloc[0:2,1])

print('\n')

#loc
print(myvar.loc[:,['Bike Category']])
```

```
0      GSA 1250
1    Speed_Twin
Name: Bike Names, dtype: object


  Bike Category
0     Adventure
1  Street Bikes
2       Touring
3        Sports
```

Pandas Series

one-dimensional, labeled array capable of holding data of any type (integer, string, float, Python objects, etc.). It can be thought of as a column in a spreadsheet or a single column of a Pandas DataFrame.

```
list_d=[1,2,3,4,5,6]
list_series=pd.Series(list_d)
print('List Series \n',list_series)
print('\n')

dict_d={1:'A',2:'B',3:'C'}
dict_series=pd.Series(dict_d)
print('Dictionary Series \n',dict_series)
print('\n')

#custom Index
data=['a','b','c','d','e']
scalar_series=pd.Series(data,index=[1,2,3,4,5])
print('Scalar Series \n',scalar_series)
print('\n')
```

```
List Series
 0    1
1    2
2    3
3    4
4    5
5    6
dtype: int64


Dictionary Series
 1    A
2    B
3    C
dtype: object


Scalar Series
 1    a
2    b
3    c
4    d
5    e
dtype: object
```

```
# Load files into a DataFrame
#read a csv file as a data frame df

df=pd.read_csv("/content/drive/MyDrive/FSDS @Kodi Senapati/Datasets/Pandas.csv")

df.head()
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Below code prints all the dataset

#print(df.to_string())


# Check systems maximum rows

print(pd.options.display.max_rows)
```

```
60
```

```
#find the columns
print('Columns:\n',df.columns)
print('\n')
#length of data frame
print('Length:\n',len(df))
print('\n')
#find shape of df
print('Shape:\n',df.shape)
```

```
Columns:
 Index(['Duration', 'Date', 'Pulse', 'Maxpulse', 'Calories'], dtype='object')


Length:
 32


Shape:
 (32, 5)
```

```
df.head()
#defaultly prints 5 rows
# You can print any number of rows by mentioning the number
# df.head(10)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0

```
# Similarly Tail

df.tail()

# You can print any number of rows by mentioning the number
# df.tail(10)
```

	Duration	Date	Pulse	Maxpulse	Calories
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	243.0

```
#The info() method prints information about the DataFrame.
#The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in eac

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  32 non-null     int64
 1   Date      31 non-null     object
 2   Pulse     32 non-null     int64
 3   Maxpulse  32 non-null     int64
 4   Calories  30 non-null     float64
dtypes: float64(1), int64(3), object(1)
memory usage: 1.4+ KB
```

```
#find if any null values in df
print(df.isnull())
```

```
    Duration   Date  Pulse  Maxpulse  Calories
0      False  False  False     False     False
1      False  False  False     False     False
2      False  False  False     False     False
3      False  False  False     False     False
4      False  False  False     False     False
5      False  False  False     False     False
6      False  False  False     False     False
7      False  False  False     False     False
8      False  False  False     False     False
9      False  False  False     False     False
10     False  False  False     False     False
11     False  False  False     False     False
12     False  False  False     False     False
13     False  False  False     False     False
14     False  False  False     False     False
15     False  False  False     False     False
16     False  False  False     False     False
17     False  False  False     False     False
18     False  False  False     False      True
19     False  False  False     False     False
20     False  False  False     False     False
21     False  False  False     False     False
22     False   True  False     False     False
23     False  False  False     False     False
24     False  False  False     False     False
25     False  False  False     False     False
26     False  False  False     False     False
27     False  False  False     False     False
28     False  False  False     False      True
29     False  False  False     False     False
30     False  False  False     False     False
31     False  False  False     False     False
```

```
#By above one we cant find exact null value.
#Summarize the count of null values column wise

print(df.isnull().sum())
```

```
Duration    0
Date        1
Pulse       0
Maxpulse    0
Calories    2
dtype: int64
```

```
df.describe()

#Gives overview of data of numbered columns
```

	Duration	Pulse	Maxpulse	Calories
count	32.000000	32.000000	32.000000	30.000000
mean	68.437500	103.500000	128.500000	304.680000
std	70.039591	7.832933	12.998759	66.003779
min	30.000000	90.000000	101.000000	195.100000
25%	60.000000	100.000000	120.000000	250.700000
50%	60.000000	102.500000	127.500000	291.200000
75%	60.000000	106.500000	132.250000	343.975000
max	450.000000	130.000000	175.000000	479.000000

```
# Slicing
df[1:10:3]

# 1 to 10 with step 3
```

	Duration	Date	Pulse	Maxpulse	Calories
1	60	'2020/12/02'	117	145	479.0
4	45	'2020/12/05'	117	148	406.0
7	450	'2020/12/08'	104	134	253.3

```
# Get column wise details

print(df[['Duration','Pulse','Maxpulse']])
```

```
    Duration  Pulse  Maxpulse
0         60    110       130
1         60    117       145
2         60    103       135
3         45    109       175
4         45    117       148
5         60    102       127
6         60    110       136
7        450    104       134
8         30    109       133
9         60     98       124
10        60    103       147
11        60    100       120
12        60    100       120
13        60    106       128
14        60    104       132
15        60     98       123
16        60     98       120
17        60    100       120
18        45     90       112
19        60    103       123
20        45     97       125
21        60    108       131
22        45    100       119
23        60    130       101
24        45    105       132
25        60    102       126
26        60    100       120
27        60     92       118
28        60    103       132
29        60    100       132
30        60    102       129
31        60     92       115
```

```
# Find data types of each column

print(df.dtypes)
```

```
Duration      int64
Date         object
Pulse         int64
Maxpulse      int64
```

```
Calories    float64
dtype: object
```

```
# Find the count of columns

print(len(df.dtypes))

print('\n')

print(len(df.columns))
```

```
5


5
```

```
#split the categorial and number columns

print(df.info())

df_int=df.select_dtypes(include=['int64'])
df_float=df.select_dtypes(include=['float64'])
df_object=df.select_dtypes(include=['object'])

print('\n')
print(len(df.columns))
print('\n')
print(len(df_int.columns))
print(len(df_float.columns))
print(len(df_object.columns))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  32 non-null     int64
 1   Date      31 non-null     object
 2   Pulse     32 non-null     int64
 3   Maxpulse  32 non-null     int64
 4   Calories  30 non-null     float64
dtypes: float64(1), int64(3), object(1)
memory usage: 1.4+ KB
None


5


3
1
1
```

Data Cleaning

which means fixing bad data into clean data.

Bad data could be:

- Empty Cell
- Data in wrong format
- Wrong Data
- Duplicates

Cleaning Empty Cells

```
df_new=df

print(df_new.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
```

```
 ---  ------    --------------  -----
 0   Duration  32 non-null     int64
 1   Date      31 non-null     object
 2   Pulse     32 non-null     int64
 3   Maxpulse  32 non-null     int64
 4   Calories  30 non-null     float64
dtypes: float64(1), int64(3), object(1)
memory usage: 1.4+ KB
None
```

```
df_new.isnull().sum()
```

	0
Duration	0
Date	1
Pulse	0
Maxpulse	0
Calories	2

df.dropna()

- this method returns a new DataFrame, and will not change the original.

df.dropna(inplace=True)

- this method will NOT return a new DataFrame, but it will remove all rows containing NULL values from the original DataFrame.

df.fillna(100,inplace=True)

- this method allows us to replace empty cells with a value

df["Duration"].fillna(100,inplace=True)

- Replace NULL values in the "Duration" columns with the number 100.

df["Calories"].fillna(df["Calories"].mean(), inplace=True)

- Replace NULL values in the "Calories" column with the mean value of that column.

df["Calories"].fillna(df["Calories"].median(), inplace=True)

- Replace NULL values in the "Calories" column with the median value of that column.

df["Calories"].fillna(df["Calories"].mode(), inplace=True)

- Replace NULL values in the "Calories" column with the mode value of that column.

```
# Drop rows if any column value is NULL
df_c=df.dropna()
# The above one returns a new DataFrame without changing the original dataFrame

print('df_c:\n',df_c.isnull().sum())
print("\n")
print('df:\n',df.isnull().sum())

#df.dropna(inplace=True)
# The above one returns a new DataFrame changing the original dataFrame
```

```
df_c:
 Duration    0
Date        0
Pulse       0
Maxpulse    0
Calories    0
dtype: int64


df:
 Duration    0
Date        1
Pulse       0
Maxpulse    0
Calories    2
```

```
dtype: int64
```

```
#
df_new["Calories"].fillna(df_new["Calories"].mean(), inplace=True)
```

```
<ipython-input-85-33da02143b9d>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df_new["Calories"].fillna(df_new["Calories"].mean(), inplace=True)
```

```
print(df_new.isnull().sum())
```

```
Duration    0
Date        1
Pulse       0
Maxpulse    0
Calories    0
dtype: int64
```

Convert into a Correct Format

```
# convert all cells in the 'Date' column into dates
df_new['Date']=pd.to_datetime(df_new['Date'],errors='coerce')
df_new['Date'].fillna(df['Date'].mean(),inplace=True)

print(df_new.isnull().sum())
```

```
Duration    0
Date        0
Pulse       0
Maxpulse    0
Calories    0
dtype: int64
<ipython-input-89-135ecb763a5e>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df_new['Date'].fillna(df['Date'].mean(),inplace=True)
```

```
# Drop duplicate rows

df_new.drop_duplicates()

print(df_new.shape)
```

```
(32, 5)
```

Finding Relationships

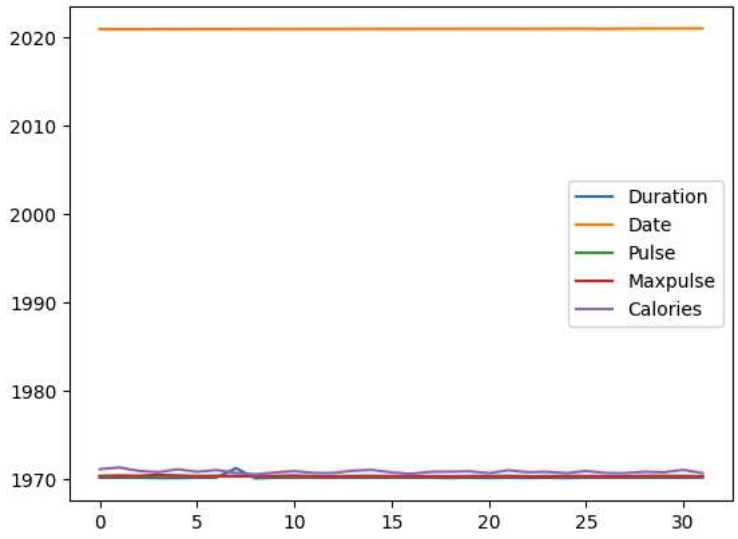
- The corr() method calculates the relationship between each column in your data set.
- The corr() method ignores "not numeric" columns.
- *Perfect Correlation:* We can see that "Duration" and "Duration" got the number 1.000000, which makes sense, each column always has a perfect relationship with itself.
- *Good Correlation:* "Duration" and "Calories" got a 0.922721 correlation, which is a very good correlation, and we can predict that the longer you work out, the more calories you burn, and the other way around: if you burned a lot of calories, you probably had a long work out.
- *Bad Correlation:* "Duration" and "Maxpulse" got a 0.009403 correlation, which is a very bad correlation, meaning that we can not predict the max pulse by just looking at the duration of the work out, and vice versa.

```
# Data Correlations

df_new.corr()
```

	Duration	Date	Pulse	Maxpulse	Calories
Duration	1.000000	-0.136839	0.004410	0.049959	-0.113923
Date	-0.136839	1.000000	-0.349479	-0.483176	-0.313595
Pulse	0.004410	-0.349479	1.000000	0.276583	0.487017
Maxpulse	0.049959	-0.483176	0.276583	1.000000	0.347419
Calories	-0.113923	-0.313595	0.487017	0.347419	1.000000

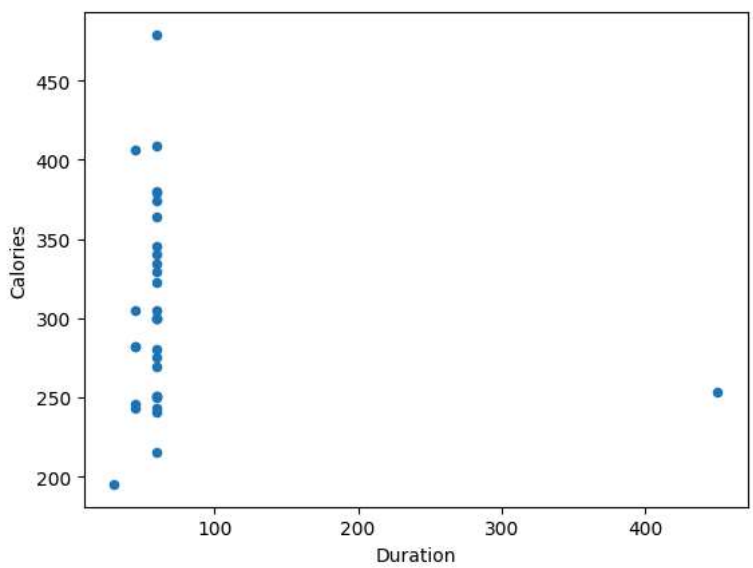
Plotting

```
import matplotlib.pyplot as plt

df_new.plot()
plt.show()
```

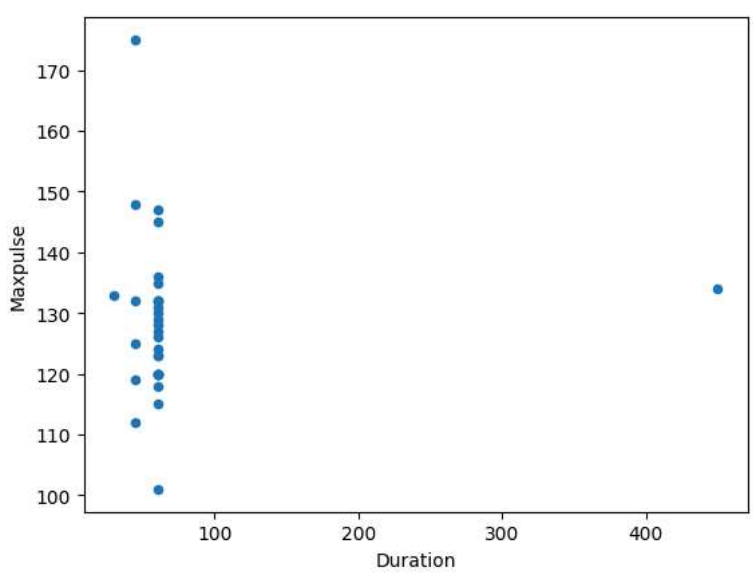


Scatter Plot

```
df_new.plot(kind='scatter',x='Duration',y='Calories')
plt.show()
```




```
df_new.plot(kind='scatter',x='Duration',y='Maxpulse')
plt.show()
```



Histogram

```
df['Calories'].plot(kind='hist')
plt.show()
```