About This Sheet
This notebook serves as a **beginner-friendly guide** to mastering the basics of Pandas. You'll learn how to:

1. Load data into Pandas.
2. Explore and analyze datasets.
3. Clean, transform and visualize data.
4. Save your work for future use.

By the end of this sheet, you'll have a solid foundation in Pandas to work on your data analysis projects. 🚀

What is Pandas? Pandas is a powerful Python library for data manipulation and analysis. It provides easy-to-use tools to handle structured data such as tables, making it an essential tool for data scientists and analysts.

With Pandas, you can:

- Load and explore datasets effortlessly.
- Clean and preprocess messy data.
- Perform statistical analysis.
- Visualize data.

Let's Go 💥

- ## Basics

1. Installation

In [1]:
```
!pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\eng mariam skoot\anaconda3\lib\site-packages (2.2.3)
Requirement already satisfied: numpy>=1.23.2 in c:\users\eng mariam skoot\anaconda3\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\eng mariam skoot\anaconda3\lib\site-packages (from pandas)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\eng mariam skoot\anaconda3\lib\site-packages (from pandas) (2023.3.post
1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\eng mariam skoot\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\eng mariam skoot\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->
pandas) (1.16.0)
```

2. Importing pandas

In [2]:
```
import pandas as pd        # pd is alternate name for referring to pandas
```

3. Pandas Series

A Pandas Series is a one-dimensional array holding data of any type (a column in a table)

In [3]:
```
a = [10,20,30]
myvar = pd.Series(a)
print(myvar)
```

```
0    10
1    20
2    30
dtype: int64
```

In [4]:
```
# The values are labeled with their index number
a = [10,20,30]
print(myvar[0])
```

```
10
```

In [5]:
```
#With the index argument, you can name your own labels.
a = [10,20,30]
myvar = pd.Series(a, index = ["i", "ii", "iii"])
print(myvar)
```

```
i      10
ii     20
iii    30
dtype: int64
```

In [6]:

```python
# You can create a Pandas Series from a dictionary

data = {'Name':['Mohamed', 'Youssef', 'Amira', 'Mariam'],
        'Age':[25, 10, 37, 19]}
ds = pd.Series(data)
print(ds)
```

```
Name    [Mohamed, Youssef, Amira, Mariam]
Age                      [25, 10, 37, 19]
dtype: object
```

Pandas DataFrames

A Pandas DataFrame is a 2 dimensional data structure (table)

In [7]:

```python
data = {'Name':['Mohamed', 'Youssef', 'Amira', 'Mariam'],
        'Age':[25, 10, 37, 19]}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age
0  Mohamed   25
1  Youssef   10
2    Amira   37
3   Mariam   19
```

In [8]:

```python
# Column Selection ==> By []
data = {'Name':['Mohamed', 'Youssef', 'Amira', 'Mariam'],
        'Age':[25, 10, 37, 19]}
df = pd.DataFrame(data)
print(df[['Age']])
```

```
   Age
0   25
1   10
2   37
3   19
```

In [9]:

```python
# Row Selection ==> By loc[]
# Note : loc[] returns pandas series
data = {'Name':['Mohamed', 'Youssef', 'Amira', 'Mariam'],
        'Age':[25, 10, 37, 19]}
df = pd.DataFrame(data)
print(df.loc[2])
```

```
Name     Amira
Age         37
Name: 2, dtype: object
```

In [10]:

```python
# Note : loc[[]] the result is a Pandas DataFrame
data = {'Name':['Mohamed', 'Youssef', 'Amira', 'Mariam'],
        'Age':[25, 10, 37, 19]}
df = pd.DataFrame(data)
print(df.loc[[1,2,3]])
```

```
      Name  Age
1  Youssef   10
2    Amira   37
3   Mariam   19
```

Read CSV Files

CSV files (comma separated files) is a plain text file used to store tabular data, where each line represents a row, and each value in the row is separated by a comma

Example:

Name, Age, City

Alice, 25, New York

Bob, 30, Los Angeles

Charlie, 35, Chicago go

data = pd.read_csv('data.csv') # 'data.csv' => this is the path of file print(data)

In [11]:
```
url = "https://media.geeksforgeeks.org/wp-content/uploads/nba.csv"
data= pd.read_csv(url)
```

Analyzing DataFrames

In [12]:
```
# The head() method returns the headers (starting from the top)
print(data.head())
```

```
            Name            Team  Number Position   Age Height  Weight  \
0  Avery Bradley  Boston Celtics     0.0       PG  25.0    6-2   180.0
1    Jae Crowder  Boston Celtics    99.0       SF  25.0    6-6   235.0
2   John Holland  Boston Celtics    30.0       SG  27.0    6-5   205.0
3    R.J. Hunter  Boston Celtics    28.0       SG  22.0    6-5   185.0
4  Jonas Jerebko  Boston Celtics     8.0       PF  29.0   6-10   231.0

             College     Salary
0              Texas  7730337.0
1          Marquette  6796117.0
2  Boston University        NaN
3      Georgia State  1148640.0
4                NaN  5000000.0
```

In [13]:
```
# The tail() method returns the headers (starting from the bottom)
print(data.tail())
```

```
             Name       Team  Number Position   Age Height  Weight College  \
453  Shelvin Mack  Utah Jazz     8.0       PG  26.0    6-3   203.0  Butler
454     Raul Neto  Utah Jazz    25.0       PG  24.0    6-1   179.0     NaN
455  Tibor Pleiss  Utah Jazz    21.0        C  26.0    7-3   256.0     NaN
456   Jeff Withey  Utah Jazz    24.0        C  26.0    7-0   231.0  Kansas
457           NaN        NaN     NaN      NaN   NaN    NaN     NaN     NaN

        Salary
453  2433333.0
454   900000.0
455  2900000.0
456   947276.0
457        NaN
```

In [14]:
```python
# info() gives you more information about the data set
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 458 entries, 0 to 457
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Name      457 non-null    object
 1   Team      457 non-null    object
 2   Number    457 non-null    float64
 3   Position  457 non-null    object
 4   Age       457 non-null    float64
 5   Height    457 non-null    object
 6   Weight    457 non-null    float64
 7   College   373 non-null    object
 8   Salary    446 non-null    float64
dtypes: float64(4), object(5)
memory usage: 32.3+ KB
None
```

In [15]:
```python
# describe() generates a statistical summary.
data.describe()
```

Out[15]:

	Number	Age	Weight	Salary
count	457.000000	457.000000	457.000000	4.460000e+02
mean	17.678337	26.938731	221.522976	4.842684e+06
std	15.966090	4.404016	26.368343	5.229238e+06
min	0.000000	19.000000	161.000000	3.088800e+04
25%	5.000000	24.000000	200.000000	1.044792e+06
50%	13.000000	26.000000	220.000000	2.839073e+06
75%	25.000000	30.000000	240.000000	6.500000e+06
max	99.000000	40.000000	307.000000	2.500000e+07

- # Manipulating Data

It's an essential step before performing data analysis or modeling, ensuring that the data is organized and clean.

1. Empty cells
2. Data in wrong format
3. Remove Duplicates
4. Filtering Data
5. Sorting
6. Adding New Columns
7. Grouping
8. Merging

1. Empty cells

How to deal with empty cells ?

- ## Remove Rows

In [16]:
```
url = "https://media.geeksforgeeks.org/wp-content/uploads/nba.csv"
data= pd.read_csv(url)
data
```

Out[16]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
...	...	...	...	...	...	...	...	...	...
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

458 rows × 9 columns

In [17]:
```
data.dropna()          # dropna() returns a new DataFrame, and will not change the original
```

Out[17]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
...	...	...	...	...	...	...	...	...	...
449	Rodney Hood	Utah Jazz	5.0	SG	23.0	6-8	206.0	Duke	1348440.0
451	Chris Johnson	Utah Jazz	23.0	SF	26.0	6-6	206.0	Dayton	981348.0
452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0

364 rows × 9 columns

If you want to change the original DataFrame: data.dropna(inplace = True)

- **Replace Empty Values**

In [18]:
```
url = "https://media.geeksforgeeks.org/wp-content/uploads/nba.csv"
data= pd.read_csv(url)
data
```

Out[18]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
...	...	...	...	...	...	...	...	...	...
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

458 rows × 9 columns

In [19]:

```
data.fillna(130)            # Replace NULL values with the number 130:
```

Out[19]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	130.0
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	130	5000000.0
...	...	...	...	...	...	...	...	...	...
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	130	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	130	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0
457	130	130	130.0	130	130.0	130	130.0	130	130.0

458 rows × 9 columns

- ### Replace Using Mean, Median, or Mode

-> Mean = the average value (the sum of all values divided by number of values).

-> Median = the value in the middle, after you have sorted all values ascending.

-> Mode = the value that appears most frequently.

In [20]:

```
url = "https://media.geeksforgeeks.org/wp-content/uploads/nba.csv"
data= pd.read_csv(url)
data
```

Out[20]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
...	...	...	...	...	...	...	...	...	...
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

458 rows × 9 columns

In [21]:

```
data['Salary'].fillna(data['Salary'].mean())        # Note the column with index 2 and 457

# data['Salary'] => because in this way the data must be a number
```

Out[21]:
```
0      7.730337e+06
1      6.796117e+06
2      4.842684e+06
3      1.148640e+06
4      5.000000e+06
           ...
453    2.433333e+06
454    9.000000e+05
455    2.900000e+06
456    9.472760e+05
457    4.842684e+06
Name: Salary, Length: 458, dtype: float64
```

2. Wrong Format

To fix wrong format, you have two options: remove the rows, or convert all cells in the columns into the same format.

- Removing Rows

In [22]:
```
data.dropna()
```

Out[22]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
...	...	...	...	...	...	...	...	...	...
449	Rodney Hood	Utah Jazz	5.0	SG	23.0	6-8	206.0	Duke	1348440.0
451	Chris Johnson	Utah Jazz	23.0	SF	26.0	6-6	206.0	Dayton	981348.0
452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0

364 rows × 9 columns

- ### Convert Into a Correct Format

  This appears in the dates

In [23]:

```
df = pd.DataFrame({'Date': {0: '26/1/2016', 1: '26/1/2016'}})
print (df)
```

```
        Date
0  26/1/2016
1  26/1/2016
```

In [24]:

```
df['Date'] = pd.to_datetime(df.Date)
print (df)
# 26/1/2016  =>  2016-01-26
```

```
        Date
0 2016-01-26
1 2016-01-26
```

```
C:\Users\Eng Mariam skoot\AppData\Local\Temp\ipykernel_22272\1954211441.py:1: UserWarning: Parsing dates in %d/%m/%Y format whe
n dayfirst=False (the default) was specified. Pass `dayfirst=True` or specify a format to silence this warning.
  df['Date'] = pd.to_datetime(df.Date)
```

- ## Converting Column type

1. .astype() is useful for explicit type conversion in pandas.
2. Use errors = 'coerce' with pd.to_numeric(), pd.to_datetime(), or pd.to_timedelta() if you need to handle errors by replacing invalid entries with NaN.

In [25]:
```
# Create a DataFrame with columns of different types
data = {
        'String_to_Int': ['10', '20', '30', '40'],          # Strings to be converted to integers
        'String_to_Float': ['1.5', '2.5', '3.5', '4.5'],    # Strings to be converted to floats
        'Category_Column': ['A', 'B', 'A', 'C'],            # Values to be converted to categorical
        'Numeric_to_Bool': [1, 0, 1, 0],                    # Numeric values to be converted to boolean
        'Invalid_to_Coerce': ['100', 'abc', '200', '300']   # Values with invalid entries for coercion
       }

df = pd.DataFrame(data)

# Print the results
print(df)
print("\nData Types:")
print(df.dtypes)          # Focus on the data types
```

```
  String_to_Int String_to_Float Category_Column  Numeric_to_Bool  \
0            10             1.5               A                1
1            20             2.5               B                0
2            30             3.5               A                1
3            40             4.5               C                0

  Invalid_to_Coerce
0               100
1               abc
2               200
3               300

Data Types:
String_to_Int        object
String_to_Float      object
Category_Column      object
Numeric_to_Bool       int64
Invalid_to_Coerce    object
dtype: object
```

In [26]:

```python
# Convert strings to integers
df['String_to_Int'] = df['String_to_Int'].astype(int)

# Convert strings to floats
df['String_to_Float'] = df['String_to_Float'].astype(float)

# Convert to categorical type
df['Category_Column'] = df['Category_Column'].astype('category')

# Convert numeric values to boolean
df['Numeric_to_Bool'] = df['Numeric_to_Bool'].astype(bool)

# Handle invalid values and convert strings to integers with errors='coerce'
df['Invalid_to_Coerce'] = pd.to_numeric(df['Invalid_to_Coerce'], errors='coerce')

# Print the results
print(df)
print("\nData Types:")
print(df.dtypes)           # Focus on the data types
```

```
   String_to_Int  String_to_Float Category_Column  Numeric_to_Bool  \
0             10              1.5               A             True   
1             20              2.5               B            False   
2             30              3.5               A             True   
3             40              4.5               C            False   

   Invalid_to_Coerce  
0              100.0  
1                NaN  
2              200.0  
3              300.0  

Data Types:
String_to_Int           int32
String_to_Float       float64
Category_Column      category
Numeric_to_Bool          bool
Invalid_to_Coerce     float64
dtype: object
```

3. Removing Duplicates

In [27]:

```
data = {
    'A': [1, 2, 2, 3, 3, 3],
    'B': ['a', 'b', 'b', 'c', 'c', 'c']
    }
df = pd.DataFrame(data)
df
```

Out[27]:

	A	B
0	1	a
1	2	b
2	2	b
3	3	c
4	3	c
5	3	c

In [28]:
```
df.duplicated()
```

Out[28]:
```
0    False
1    False
2     True
3    False
4     True
5     True
dtype: bool
```

In [29]:
```
df.duplicated().sum()          # duplicated().sum() => summision of duplicated values
```

Out[29]:
```
3
```

In [30]:
```
# Remove duplicate rows
df.drop_duplicates(inplace = True)
df
```

Out[30]:

	A	B
0	1	a
1	2	b
3	3	c

In [31]:
```
df.duplicated().sum()
```

Out[31]:
```
0
```

4. Filtering Data

In [32]:
```
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [24, 27, 22, 32],
        'Score': [85, 62, 90, 70]
       }
df = pd.DataFrame(data)

# Filter rows where Age is greater than 25
filtered_df = df[df['Age'] > 25]
filtered_df
```

Out[32]:

	Name	Age	Score
1	Bob	27	62
3	David	32	70

5. Sorting

In [33]:
```
df = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 22, 35]
    })

# Sort the DataFrame by the 'Age' column
sorted_df = df.sort_values(by='Age')
sorted_df
```

Out[33]:

	Name	Age
2	Charlie	22
0	Alice	25
1	Bob	30
3	David	35

6. Adding New Columns

In [34]:

```
data = {
    'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
    'Height': [5.1, 6.2, 5.1, 5.2],
    'Qualification': ['Msc', 'MA', 'Msc', 'Msc']
       }
df = pd.DataFrame(data)
df
```

Out[34]:

	Name	Height	Qualification
0	Jai	5.1	Msc
1	Princi	6.2	MA
2	Gaurav	5.1	Msc
3	Anuj	5.2	Msc

In [35]:

```
address = ['NewYork', 'Chicago', 'Boston', 'Miami']
df['Address'] = address      # Adding the column
df
```

Out[35]:

	Name	Height	Qualification	Address
0	Jai	5.1	Msc	NewYork
1	Princi	6.2	MA	Chicago
2	Gaurav	5.1	Msc	Boston
3	Anuj	5.2	Msc	Miami

7. Grouping

In [36]:

```
data = {
    'state': ['CA', 'NY', 'CA', 'NY', 'CA'],
    'value': [1, 2, 3, 4, 5]
    }
df = pd.DataFrame(data)

# Group by 'state'
grouped = df.groupby('state')

# Apply a function to each group
result = grouped.sum()
result
```

Out[36]:

	value
state	
CA	9
NY	6

8. Merging

In [37]:

```
data1 = {
        'key': ['K0', 'K1', 'K2', 'K3'],
        'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
```

```
        'Age': [27, 24, 22, 32]
        }
data2 = {
        'key': ['K0', 'K1', 'K2', 'K3'],
         'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
         'Qualification': ['Btech', 'B.A', 'Bcom', 'B.hons']
        }

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
print(df1,"\n\n",df2)
```

```
  key    Name  Age
0  K0     Jai   27
1  K1  Princi   24
2  K2  Gaurav   22
3  K3    Anuj   32

   key    Address Qualification
0  K0     Nagpur         Btech
1  K1     Kanpur           B.A
2  K2  Allahabad          Bcom
3  K3    Kannuaj        B.hons
```

In [38]:

```
# Merge DataFrames on the 'key' column
result = pd.merge(df1, df2, on='key')         # on => 2-DataFrames have the same column named 'Key'
print(result)
```

```
  key    Name  Age    Address Qualification
0  K0     Jai   27     Nagpur         Btech
1  K1  Princi   24     Kanpur           B.A
2  K2  Gaurav   22  Allahabad          Bcom
3  K3    Anuj   32    Kannuaj        B.hons
```

- # Advanced

1. Correlations

2. Plotting (Data visualization)

1. Correlations

The corr() method calculates the relationship between each column in your data set.

In [39]:
```
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"
# read dataset
df = pd.read_csv(url, header=None)

## Set column names instead of being in index
df.columns = ['Pregnancies', 'Glucose', 'BloodPressure',
              'SkinThickness', 'Insulin', 'BMI',
              'DiabetesPedigreeFunction', 'Age', 'Outcome']

df.head()
```

Out[39]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [40]:
```
# Show the relationship between the columns:
df.corr()
```

Out[40]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356

2. Data visualization

We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen.

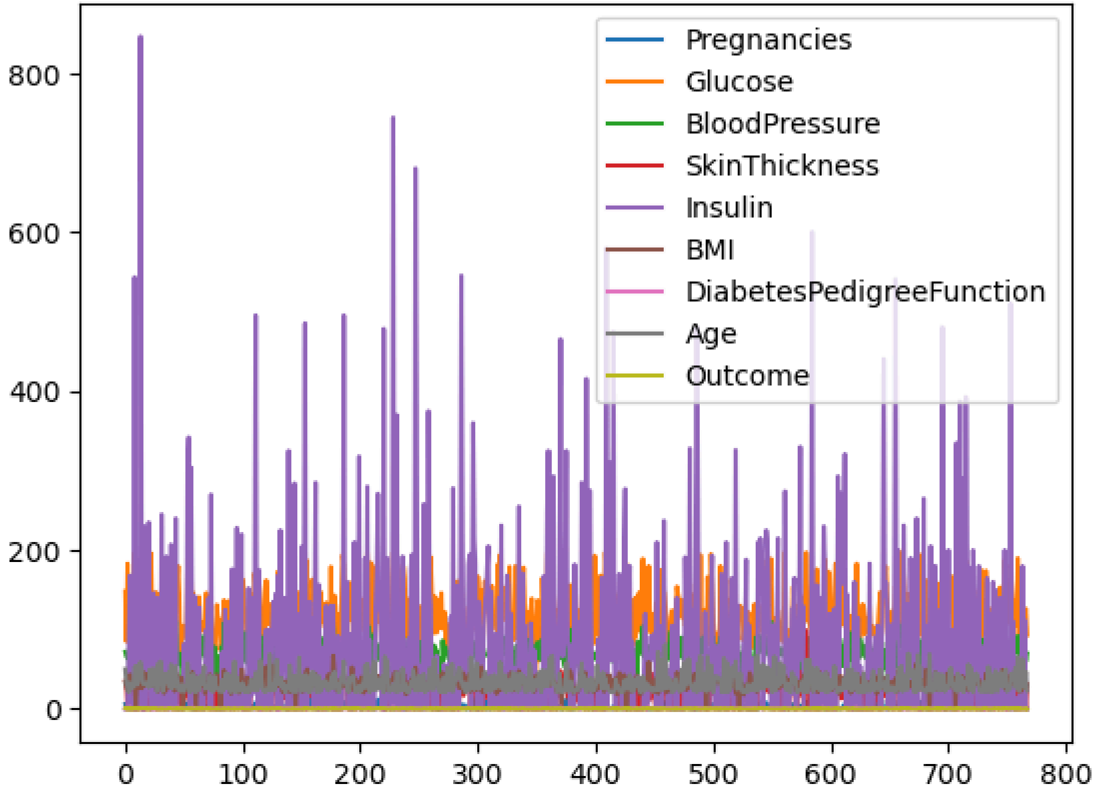
To install matplotlib :

!pip install matplotlib

In [41]:

```
import pandas as pd
import matplotlib.pyplot as plt

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"
df = pd.read_csv(url, header=None)            # read dataset

## Set column names instead of being in index
df.columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
              'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

df.plot()           # visualize the DataFrame
```

```
plt.show()
```



In [42]:
```
df.head()
```

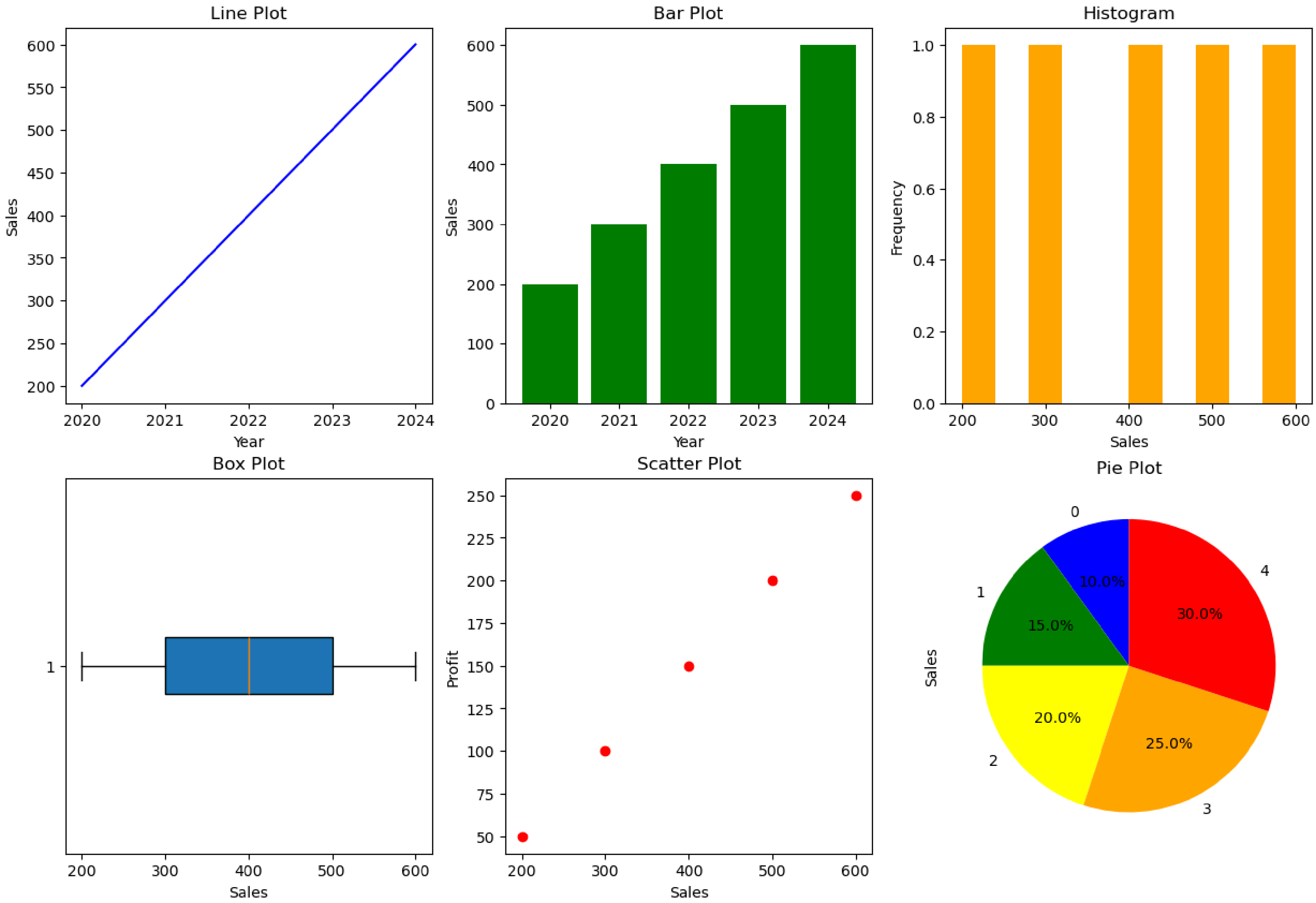
Out[42]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [43]:

```python
data = {
    "Year": [2020, 2021, 2022, 2023, 2024],
    "Sales": [200, 300, 400, 500, 600],
    "Profit": [50, 100, 150, 200, 250]
}

df = pd.DataFrame(data)

plt.figure(figsize=(15, 10))              # Figure setting with 2 axes


# plt.subplot(2, 3, 1)
# 2: The grid will have 2 rows.
# 3: The grid will have 3 columns.
# 1: Which position the current plot will occupy in the grid?

#  Line Plot
plt.subplot(2, 3, 1)
plt.plot(df['Year'], df['Sales'], color='blue')         # x-axis as "Year" , y-axis as "Sales"
plt.title("Line Plot")                # title to the graph
plt.xlabel("Year")
plt.ylabel("Sales")

#  Bar Plot
plt.subplot(2, 3, 2)
plt.bar(df['Year'], df['Sales'], color='green')        # x-axis as "Year" , y-axis as "Sales"
plt.title("Bar Plot")                # title to the graph
plt.xlabel("Year")
```

```
plt.ylabel("Sales")

#  Histogram
plt.subplot(2, 3, 3)
plt.hist(df['Sales'], color='orange')                 # histogram to show the distribution of 'Sales'
plt.title("Histogram")                # title to the graph
plt.xlabel("Sales")
plt.ylabel("Frequency")

#  Box Plot
plt.subplot(2, 3, 4)
plt.boxplot(df['Sales'], vert=False, patch_artist=True)    # Create a box plot of 'Sales' with horizontal orientation and colo
plt.title("Box Plot")
plt.xlabel("Sales")

#  Scatter Plot
plt.subplot(2, 3, 5)
plt.scatter(df['Sales'], df['Profit'], color='red')        # x-axis as "Year" , y-axis as "Sales"
plt.title("Scatter Plot")
plt.xlabel("Sales")
plt.ylabel("Profit")

#  Pie Plot
plt.subplot(2, 3, 6)
# Create a pie chart of 'Sales' with percentage values, specific colors, and starting angle of 90 degrees.
df['Sales'].plot(kind='pie', autopct='%1.1f%%', colors=['blue', 'green', 'yellow', 'orange', 'red'], startangle=90)
plt.title("Pie Plot")

# What is (autopct='%1.1f%%') ?
 # autopct: Controls the display of percentages on the pie slices ()
 # '%1.1f%%': Formats percentages with one decimal place followed by a % symbol.


plt.show()              # Display all the plots in the 2x3 grid.
```


Line Plot
Sales
600
550
500
450
400
350
300
250
200
2020
2021
2022
2023
2024
Year
Bar Plot
Sales
600
500
400
300
200
100
0
2020
2021
2022
2023
2024
Year
Histogram
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
200
300
400
500
600
Sales
Box Plot
1
200
300
400
500
600
Sales
Scatter Plot
Profit
250
225
200
175
150
125
100
75
50
200
300
400
500
600
Sales
Pie Plot
Sales
0
1
2
3
4
10.0%
15.0%
20.0%
25.0%
30.0%

Saving Data

In [44]:
```
#Saving Data as csv file
df.to_csv("output.csv", index=False)
```

In [45]:
```
# Let's try it
data= pd.read_csv("output.csv")
data.head()             # it works 🤩
```

Out[45]:

	Year	Sales	Profit
0	2020	200	50
1	2021	300	100
2	2022	400	150
3	2023	500	200
4	2024	600	250

I hope you found this notebook helpful! 😊 These are the key concepts we covered today:

- **Pandas Basics**
- **Data Manipulation and Cleaning**
- **Data Visualization**
- **Exporting Processed Data**

Data is the new gold, and your skills now make you able to extract their value! 🚀