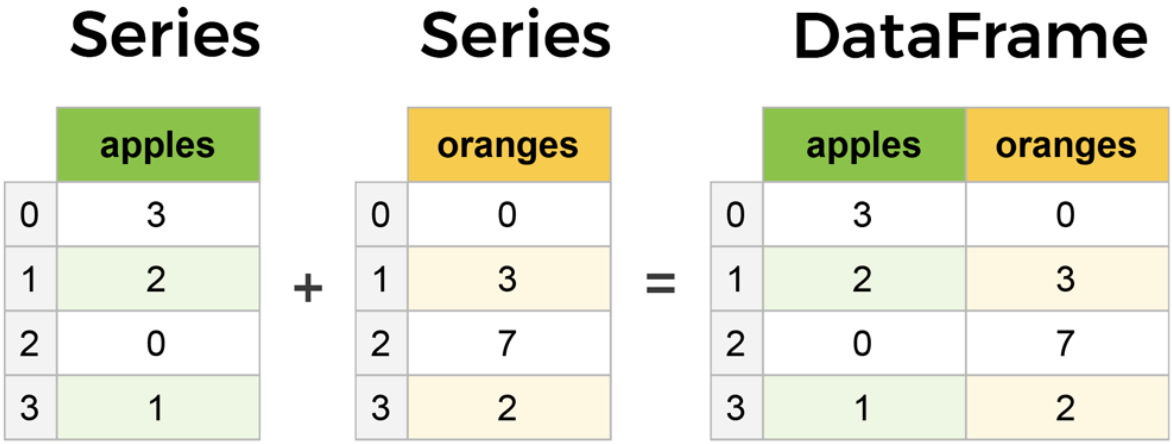
Pandas

Pandas is a Python library. Pandas is used to analyze data. pandas is a data manipulation package in Python for tabular data. That is, data in the form of rows and columns, also known as DataFrames. You can think of a DataFrame as an Excel sheet.



What is pandas used for?

-> pandas is used throughout the data analysis workflow. With pandas, you can: -> Import datasets from databases, spreadsheets, comma-separated values (CSV) files, and more. -> Clean datasets, for example, by dealing with missing values. -> Tidy datasets by reshaping their structure into a suitable format for analysis. -> Aggregate data by calculating summary statistics such as the mean of columns, correlation between them, and more.

Install pandas

In [2]:
```
#Installing pandas is straightforward; just use the pip install command in your termin
!pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\teks108\anaconda3\lib\site-packages
(2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\teks108\anaconda3\l
ib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\teks108\anaconda3\lib\site-pa
ckages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\teks108\anaconda3\lib\site-
packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in c:\users\teks108\anaconda3\lib\site-p
ackages (from pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in c:\users\teks108\anaconda3\lib\site-packag
es (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

Importing data in pandas

In [5]:
```
#To begin working with pandas, import the pandas Python package as shown below.
#When importing pandas, the most common alias for pandas is pd.
import pandas as pd
```

Importing CSV files

In [ ]:
```
#Use read_csv() with the path to the CSV file to read a comma-separated values file
df = pd.read_csv("diabetes.csv")  #In parathesis you need to give path of dataset
```

Importing text files

In [6]:
```
#you need to specify a separator with the sep argument, as shown below.
# The separator argument refers to the symbol used to separate rows in a DataFrame.
# Comma (sep = ","), whitespace(sep = "\s"), tab (sep = "\t"), and colon(sep = ":") ar
# Here \s represents a single white space character.

df = pd.read_csv("diabetes.txt", sep="\s")
```

Importing Excel files (single sheet)

In [ ]:
```
#Reading excel files (both XLS and XLSX) is as easy as the read_excel() function, usin
df = pd.read_excel('diabetes.xlsx')
```

Importing Excel files (multiple sheets)

Reading Excel files with multiple sheets is not that different. You just need to specify one additional argument, sheet_name, where you can either pass a string for the sheet name or an integer for the sheet position. (note that Python uses 0-indexing, where the first sheet can be accessed with sheet_name = 0)

In [ ]:
```
# Extracting the second sheet since Python uses 0-indexing
df = pd.read_excel('diabetes_multi.xlsx', sheet_name=1)
```

Importing JSON file

Similar to the read_csv() function, you can use read_json() for JSON file types with the JSON file name as the argument

In [ ]:
```
df = pd.read_json("diabetes.json")
```

Example

In [11]:
```
#loading csv file
df = pd.read_csv(r"D:\Vamsi Reddy\Dataset\50_Startups.csv")   # r stands for "raw"
```

Viewing and understanding DataFrames using pandas

How to view data using .head() and .tail()

In [12]:
```
# using head()
df.head()
```

Out[12]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

In [13]:
```
# using tail()
df.tail()
```

Out[13]:

	R&D Spend	Administration	Marketing Spend	State	Profit
45	1000.23	124153.04	1903.93	New York	64926.08
46	1315.46	115816.21	297114.46	Florida	49490.75
47	0.00	135426.92	0.00	California	42559.73
48	542.05	51743.15	0.00	New York	35673.41
49	0.00	116983.80	45173.06	California	14681.40

In [14]:
```
# The .describe() method prints the summary statistics of all numeric columns,
# such as count, mean, standard deviation, range, and quartiles of numeric columns.
df.describe()
```

Out[14]:

	R&D Spend	Administration	Marketing Spend	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

In [16]:
```
#The .info() method is a quick way to look at the data types, missing values, and data

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object 
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

In [20]:
```
print(df.shape) # Get the number of rows and columns
print(df.shape[0]) # Get the number of rows only
print(df.shape[1]) # Get the number of columns only
```

```
(50, 5)
50
5
```

In [22]:
```
#Get all columns and column names
df.columns
```

Out[22]:
```
Index(['R&D Spend', 'Administration', 'Marketing Spend', 'State', 'Profit'], dtype='object')
```

In [23]:
```
#It can be converted to a list using a list() function
list(df.columns)
```

Out[23]:
```
['R&D Spend', 'Administration', 'Marketing Spend', 'State', 'Profit']
```

Checking for missing values in pandas

In [27]:
```
df.isnull().head(10)
```

Out[27]:

	R&D Spend	Administration	Marketing Spend	State	Profit
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

In [28]:
```
# finding Null values and adding them
df.isnull().sum()
```

Out[28]:
```
R&D Spend          0
Administration     0
Marketing Spend    0
State              0
Profit             0
dtype: int64
```

Slicing and Extracting Data in pandas

In [30]:
```
#Isolating one column using [ ]
df['Profit'].head()
```

Out[30]:
```
0    192261.83
1    191792.06
2    191050.39
3    182901.99
4    166187.94
Name: Profit, dtype: float64
```

In [36]:
```
#Isolating two or more columns using [[ ]]
df[['State', 'Profit']].head()
```

Out[36]:

	State	Profit
0	New York	192261.83
1	California	191792.06
2	Florida	191050.39
3	New York	182901.99
4	Florida	166187.94

In [39]:
```
#Isolating one row using [ ]
df[df.index==2]
```

Out[39]:

	R&D Spend	Administration	Marketing Spend	State	Profit
2	153441.51	101145.55	407934.54	Florida	191050.39

In [42]:
```
#Isolating two or more rows using [ ]
#Similarly, two or more rows can be returned using the .isin() method instead of a ==
df[df.index.isin(range(2,10))]
```

Out[42]:

	R&D Spend	Administration	Marketing Spend	State	Profit
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	127716.82	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96

Using .loc[] and .iloc[] to fetch rows

You can fetch specific rows by labels or conditions using .loc[] and .iloc[] ("location" and "integer location"). .loc[] uses a label to point to a row, column or cell, whereas .iloc[] uses the numeric position.

In [46]:
```
df.loc[5]
```

Out[46]:
```
R&D Spend           131876.9
Administration       99814.71
Marketing Spend    362861.36
State               New York
Profit             156991.12
Name: 5, dtype: object
```

In [47]:
```
df.iloc[5]
```

Out[47]:
```
R&D Spend           131876.9
Administration       99814.71
Marketing Spend    362861.36
State               New York
Profit             156991.12
Name: 5, dtype: object
```

In [48]:
```
#you can also fetch multiple rows by providing a range in square brackets.
df.iloc[2:10]
```

Out[48]:

	R&D Spend	Administration	Marketing Spend	State	Profit
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	127716.82	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96

In [49]:
```
df.loc[2:10]
```

Out[49]:

	R&D Spend	Administration	Marketing Spend	State	Profit
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	127716.82	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96
10	101913.08	110594.11	229160.95	Florida	146121.95

Cleaning data using pandas

Data cleaning is one of the most common tasks in data science.

In [68]:
```
data = pd.read_csv(r"D:\Vamsi Reddy\Dataset\titanic.csv", usecols =['Age','Fare','Surv
```

In [69]:
```
data.head()
```

Out[69]:

	Survived	Age	Fare
0	0	34.5	7.8292
1	1	47.0	7.0000
2	0	62.0	9.6875
3	0	27.0	8.6625
4	1	22.0	12.2875

In [70]:
```
# finding the null values and sum them together
data.isnull().sum()
```

Out[70]:
```
Survived     0
Age         86
Fare         1
dtype: int64
```

Dealing with missing data technique

1. Dropping missing values

One way to deal with missing data is to drop it. This is particularly useful in cases where you have plenty of data and losing a small portion won't impact the downstream analysis. You can use a .dropna() method

In [71]:
```
data1 = data.dropna()
```

In [72]:
```
# finding the null values after the dropna() method
data1.isnull().sum()
```

Out[72]:
```
Survived    0
Age         0
Fare        0
dtype: int64
```

The axis argument lets you specify whether you are dropping rows, or columns, with missing values. The default axis removes the rows containing NaNs. Use axis = 1 to remove the columns with one or more NaN values. Also, notice how we are using the argument inplace=True which lets you skip saving the output of .dropna() into a new DataFrame.

In [90]:
```
data = pd.read_csv(r"D:\Vamsi Reddy\Dataset\titanic.csv", usecols =['Age','Fare','Surv
```

In [91]:
```
data.dropna(inplace=True, axis=0)
print(data.head())
print(data.isnull().sum())
```

```
   Survived   Age     Fare
0         0  34.5   7.8292
1         1  47.0   7.0000
2         0  62.0   9.6875
3         0  27.0   8.6625
4         1  22.0  12.2875
Survived    0
Age         0
Fare        0
dtype: int64
```

In [ ]:
```

```

2: Replacing missing values

In [105…
```
data = pd.read_csv(r"D:\Vamsi Reddy\Dataset\titanic.csv", usecols =['Age','Fare','Surv
data.head(12)
```

Out[105]:

	Survived	Age	Fare
0	0	34.5	7.8292
1	1	47.0	7.0000
2	0	62.0	9.6875
3	0	27.0	8.6625
4	1	22.0	12.2875
5	0	14.0	9.2250
6	1	30.0	7.6292
7	0	26.0	29.0000
8	1	18.0	7.2292
9	0	21.0	24.1500
10	0	NaN	7.8958
11	0	46.0	26.0000

In [106…

```
#replacing the NAN value usinf mean()
mean_value = data['Age'].mean()
print(mean_value)
```

```
30.272590361445783
```

In [107…

```
data3 = data.fillna(mean_value)
```

In [108…

```
data3.head(12)
```

Out[108]:

	Survived	Age	Fare
0	0	34.50000	7.8292
1	1	47.00000	7.0000
2	0	62.00000	9.6875
3	0	27.00000	8.6625
4	1	22.00000	12.2875
5	0	14.00000	9.2250
6	1	30.00000	7.6292
7	0	26.00000	29.0000
8	1	18.00000	7.2292
9	0	21.00000	24.1500
10	0	30.27259	7.8958
11	0	46.00000	26.0000

In [109…

```
data = pd.read_csv(r"D:\Vamsi Reddy\Dataset\titanic.csv", usecols =['Age','Fare','Surv
data.head(12)
```

Out[109]:

	Survived	Age	Fare
0	0	34.5	7.8292
1	1	47.0	7.0000
2	0	62.0	9.6875
3	0	27.0	8.6625
4	1	22.0	12.2875
5	0	14.0	9.2250
6	1	30.0	7.6292
7	0	26.0	29.0000
8	1	18.0	7.2292
9	0	21.0	24.1500
10	0	NaN	7.8958
11	0	46.0	26.0000

In [114…

```
#replacing the NAN value usinf median()
median_value =data['Age'].median()
data4 = data.fillna(median_value)
print(median_value)
```

```
27.0
```

In [115…

```
data4.head(12)
```

Out[115]:

	Survived	Age	Fare
0	0	34.5	7.8292
1	1	47.0	7.0000
2	0	62.0	9.6875
3	0	27.0	8.6625
4	1	22.0	12.2875
5	0	14.0	9.2250
6	1	30.0	7.6292
7	0	26.0	29.0000
8	1	18.0	7.2292
9	0	21.0	24.1500
10	0	27.0	7.8958
11	0	46.0	26.0000

3.Duplicate Data

In [120…
```
data = pd.read_csv(r"D:\Vamsi Reddy\Dataset\titanic.csv", usecols =['Age','Fare','Surv
data.head(12)
```

Out[120]:

	Survived	Age	Fare
0	0	34.5	7.8292
1	1	47.0	7.0000
2	0	62.0	9.6875
3	0	27.0	8.6625
4	1	22.0	12.2875
5	0	14.0	9.2250
6	1	30.0	7.6292
7	0	26.0	29.0000
8	1	18.0	7.2292
9	0	21.0	24.1500
10	0	NaN	7.8958
11	0	46.0	26.0000

In [121…
```
#shape of dataset
data.shape
```

Out[121]:
```
(418, 3)
```

In [122…
```
data5 = data.drop_duplicates()
data5.shape
```

Out[122]:
```
(370, 3)
```

Renaming columns

In [123…
```
data = pd.read_csv(r"D:\Vamsi Reddy\Dataset\titanic.csv", usecols =['Age','Fare','Surv
data.head()
```

Out[123]:

	Survived	Age	Fare
0	0	34.5	7.8292
1	1	47.0	7.0000
2	0	62.0	9.6875
3	0	27.0	8.6625
4	1	22.0	12.2875

In [124…
```
# renaming the column name
data.rename(columns = {'Age':'New_Age'}, inplace = True)
data.head()
```

Out[124]:

	Survived	New_Age	Fare
0	0	34.5	7.8292
1	1	47.0	7.0000
2	0	62.0	9.6875
3	0	27.0	8.6625
4	1	22.0	12.2875

Data analysis in pandas

In [125…
```
#Summary operators (mean, mode, median)
```

In [126…
```
#you can get the mean of each column value using the .mean() method.
data.mean()
```

Out[126]:
```
Survived     0.363636
New_Age     30.272590
Fare        35.627188
dtype: float64
```

In [127…
```
#A mode can be computed similarly using the .mode() method.
data.mode()
```

Out[127]:

	Survived	New_Age	Fare
0	0.0	21.0	7.75
1	NaN	24.0	NaN

In [128…
```
#the median of each column is computed with the .median() method
data.median()
```

Out[128]:
```
Survived     0.0000
New_Age     27.0000
Fare        14.4542
dtype: float64
```

count the number of observations each category has in a column. Category values can be counted using the .value_counts() methods.

In [140…
```
data = pd.read_csv(r"D:\Vamsi Reddy\Dataset\titanic.csv")
data.head()
```

Out[140]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Eml
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	

In [131…
```
data['Sex'].value_counts()
```

Out[131]:
```
Sex
male      266
female    152
Name: count, dtype: int64
```

In [ ]:
```

```

Aggregating data with .groupby() in pandas

You can do that by combining the .groupby() method with a summary method of your choice.

when you going to apply groupby() you need to clean the data

In [167…
```
data = pd.read_csv(r"D:\Vamsi Reddy\Dataset\titanic.csv", usecols =['Age','Fare','Cabi
data.head()
```

Out[167]:

	Survived	Age	Fare	Cabin
0	0	34.5	7.8292	NaN
1	1	47.0	7.0000	NaN
2	0	62.0	9.6875	NaN
3	0	27.0	8.6625	NaN
4	1	22.0	12.2875	NaN

In [168…
```
mean_value = data['Age'].mean()
print(mean_value)
```

```
30.272590361445783
```

In [169…
```
data.fillna(mean_value).head()
```

Out[169]:

	Survived	Age	Fare	Cabin
0	0	34.5	7.8292	30.27259
1	1	47.0	7.0000	30.27259
2	0	62.0	9.6875	30.27259
3	0	27.0	8.6625	30.27259
4	1	22.0	12.2875	30.27259

In [179…
```
data.groupby('Cabin').mean().head()
```

Out[179]:

	Survived	Age	Fare
Cabin			
A11	1.0	33.0	27.7208
A18	0.0	39.0	29.7000
A21	0.0	41.0	30.5000
A29	1.0	36.0	31.6792
A34	0.5	53.5	81.8583

In [ ]:
```

```

In [ ]:
```

```