Import pandas

In [1]:
```
import pandas as pd
import numpy as np
```

1. Read Data

Read csv data

A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record https://en.wikipedia.org/wiki/Comma-separated_values (https://en.wikipedia.org/wiki/Comma-separated_values). In pandas we use the pandas.read_csv() method to read csv data. The read_csv() method accepts important arguments such as filepath_or_buffer which specifies the file path. sep indicates the delimiter to use, engine determines which engine to use between C which is faster but less features or Python which is slower but feature-complete. usecols defines the columns to be fetched, nrows which specifies the number of rows to read, chunksize limits the amount or records to fetch at a time and many other arguments.

In [2]:
```
titanic_df=pd.read_csv('titanic.csv')
titanic_df.head()
```

Out[2]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500

Read Excel data

In [3]:
```
titanic_excel_df=pd.read_excel('titanic.xlsx','Sheet1')
titanic_excel_df
```

Out[3]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500
...	...	...	...	...	...	...	...	...
882	0	2	Rev. Juozas Montvila	male	27.0	0	0	13.0000
883	1	1	Miss. Margaret Edith Graham	female	19.0	0	0	30.0000
884	0	3	Miss. Catherine Helen Johnston	female	7.0	1	2	23.4500
885	1	1	Mr. Karl Howell Behr	male	26.0	0	0	30.0000
886	0	3	Mr. Patrick Dooley	male	32.0	0	0	7.7500

887 rows × 8 columns

Read html file

The HyperText Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser. https://en.wikipedia.org/wiki/HTML (https://en.wikipedia.org/wiki/HTML). Pandas has pandas.read_html() that extracts data from html files.

In [4]:
```
gdp_df=pd.read_html('https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)#Table')[2]
gdp_df.columns=['country','continent','imf_estimate','imf_year','un_estimate','un_year','wb_estimate','wb_year'] # rename columns
gdp_df.head()
```

Out[4]:

	country	continent	imf_estimate	imf_year	un_estimate	un_year	wb_estimate	wb_year
0	United States	Americas	22939580.0	2021	21433226	2019	20936600.0	2020
1	China	Asia	16862979.0	[n 2]2021	14342933	[n 3]2019	14722731.0	2020
2	Japan	Asia	5103110.0	2021	5082465	2019	4975415.0	2020
3	Germany	Europe	4230172.0	2021	3861123	2019	3806060.0	2020
4	United Kingdom	Europe	3108416.0	2021	2826441	2019	2707744.0	2020

Read Json Data

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write https://www.json.org/json-en.html (https://www.json.org/json-en.html). We can read json data to pandas dataframe using the read_json() method. pd.read_json() method converts a json data to a pandas dataframe. It accepts important arguments such as; path_or_buf which is the location of the data source file. chunksize which is useful when fetching large data. encoding decodes the data to a readable formart. orient defines the expected json formart and can take any of index, split, records, columns or values. Check the pandas documentation for more details https://pandas-docs.github.io/pandas-docs-travis/reference/api/pandas.read_json.html (https://pandas-docs.github.io/pandas-docs-travis/reference/api/pandas.read_json.html)

In [5]:
```
# json_df=pd.read_json("https://raw.githubusercontent.com/BindiChen/machine-learning/master/data-analysis/027-pandas-convert-json/data/simple.json")
json_exchange_rate=pd.read_json("https://api.exchangerate-api.com/v4/latest/USD")
json_exchange_rate=json_exchange_rate[['provider','base','date','time_last_updated','rates']].head()
json_exchange_rate
```

Out[5]:

	provider	base	date	time_last_updated	rates
AED	https://www.exchangerate-api.com	USD	2022-01-13	1642032001	3.67
AFN	https://www.exchangerate-api.com	USD	2022-01-13	1642032001	105.37
ALL	https://www.exchangerate-api.com	USD	2022-01-13	1642032001	107.13
AMD	https://www.exchangerate-api.com	USD	2022-01-13	1642032001	481.78
ANG	https://www.exchangerate-api.com	USD	2022-01-13	1642032001	1.79

Parquet

Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language https://parquet.apache.org/ (https://parquet.apache.org/). The pandas.read_parquet() method reads the parquet data file to pandas data frame. It provides a partitioned binary columnar serialization for data frames.

First we will create a parquet file and demonstrated how to read it in pandas

In [6]:
```
# Uncomment this command to install pyarrow
# !pip install pyarrow
```

In [7]:
```
titanic_df.to_parquet('parquet_sample_data.parquet', engine='pyarrow')
```

In [8]:
```
# Read parquet file
parquet_df=pd.read_parquet('parquet_sample_data.parquet', engine='pyarrow')
parquet_df.head()
```

Out[8]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500

Read Pickle file

Pickle is a file formart that serializes Python object to byte stream. In Machine Learning we use pickle files to share models and deploy them in production. Let's first create a pickle file and demonstrate how to read it in pandas.

In [9]:
```
json_exchange_rate.to_pickle('pickle_sample_data.pkl')
pickle_df=pd.read_pickle('pickle_sample_data.pkl')
pickle_df.head()
```

Out[9]:

	provider	base	date	time_last_updated	rates
AED	https://www.exchangerate-api.com	USD	2022-01-13	1642032001	3.67
AFN	https://www.exchangerate-api.com	USD	2022-01-13	1642032001	105.37
ALL	https://www.exchangerate-api.com	USD	2022-01-13	1642032001	107.13
AMD	https://www.exchangerate-api.com	USD	2022-01-13	1642032001	481.78
ANG	https://www.exchangerate-api.com	USD	2022-01-13	1642032001	1.79

Read SQL Data

SQL is a popular language for working with and manipulating data in the databases. In Data Science and Analytics understanding SQL is a key skills to efficiently and commfortably work with data. Modern databases such as Oracle, MSSQL, GBQ and Amazon Redshift have advanced to the level beyond being a data storage container to providing capabilities such as writing advanced in-database Machine Learning models through SQL. Pandas enables us to read data from the database through SQL. To read data through SQL in pandas we first need to install the respective python-database driver.

In [10]:
```
# Let's first create an sqlite database and save titanic dataset to it
from sqlalchemy import create_engine
engine = create_engine('sqlite:///:memory:')
titanic_df.to_sql('titanic_sqlite_data', engine, chunksize=100)
```

In [11]:
```
titanic_sqlite_df=pd.read_sql_table('titanic_sqlite_data', engine)
titanic_sqlite_df.head()
```

Out[11]:

	index	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500

2. Save Data

Save data to csv

In [12]:
```
json_exchange_rate.to_csv("Exchange_Rate.csv")
gdp_df.to_csv("gdp.csv",index=False)
```

Save data to excel

In [13]:
```
json_exchange_rate.to_excel("Exchange_Rate.xlsx")
```

Save data to html

In [14]:
```
titanic_df.head().to_html()
```

Out[14]:
```
'<table border="1" class="dataframe">\n  <thead>\n    <tr style="text-align: right;">\n      <th></th>\n      <th>Sur
vived</th>\n      <th>Pclass</th>\n      <th>Name</th>\n      <th>Sex</th>\n      <th>Age</th>\n      <th>Siblings/Sp
ouses Aboard</th>\n      <th>Parents/Children Aboard</th>\n      <th>Fare</th>\n    </tr>\n  </thead>\n  <tbody>\n
<tr>\n      <th>0</th>\n      <td>0</td>\n      <td>3</td>\n      <td>Mr. Owen Harris Braund</td>\n      <td>male</td
>\n      <td>22.0</td>\n      <td>1</td>\n      <td>0</td>\n      <td>7.2500</td>\n    </tr>\n    <tr>\n      <th>1</
th>\n      <td>1</td>\n      <td>1</td>\n      <td>Mrs. John Bradley (Florence Briggs Thayer) Cumings</td>\n      <td
>female</td>\n      <td>38.0</td>\n      <td>1</td>\n      <td>0</td>\n      <td>71.2833</td>\n    </tr>\n    <tr>\n
<th>2</th>\n      <td>1</td>\n      <td>3</td>\n      <td>Miss. Laina Heikkinen</td>\n      <td>female</td>\n      <t
d>26.0</td>\n      <td>0</td>\n      <td>0</td>\n      <td>7.9250</td>\n    </tr>\n    <tr>\n      <th>3</th>\n
<td>1</td>\n      <td>1</td>\n      <td>Mrs. Jacques Heath (Lily May Peel) Futrelle</td>\n      <td>female</td>\n
<td>35.0</td>\n      <td>1</td>\n      <td>0</td>\n      <td>53.1000</td>\n    </tr>\n    <tr>\n      <th>4</th>\n
<td>0</td>\n      <td>3</td>\n      <td>Mr. William Henry Allen</td>\n      <td>male</td>\n      <td>35.0</td>\n
<td>0</td>\n      <td>0</td>\n      <td>8.0500</td>\n    </tr>\n  </tbody>\n</table>'
```

Save data to json

In [15]:
```
titanic_df.head().to_json(orient='columns')
```

Out[15]:
```
'{"Survived":{"0":0,"1":1,"2":1,"3":1,"4":0},"Pclass":{"0":3,"1":1,"2":3,"3":1,"4":3},"Name":{"0":"Mr. Owen Harris Br
aund","1":"Mrs. John Bradley (Florence Briggs Thayer) Cumings","2":"Miss. Laina Heikkinen","3":"Mrs. Jacques Heath (L
ily May Peel) Futrelle","4":"Mr. William Henry Allen"},"Sex":{"0":"male","1":"female","2":"female","3":"femal
e","4":"male"},"Age":{"0":22.0,"1":38.0,"2":26.0,"3":35.0,"4":35.0},"Siblings\\/Spouses Aboard":{"0":1,"1":1,"2":
0,"3":1,"4":0},"Parents\\/Children Aboard":{"0":0,"1":0,"2":0,"3":0,"4":0},"Fare":{"0":7.25,"1":71.2833,"2":7.92
5,"3":53.1,"4":8.05}}'
```

Save data to Parquet

In [16]:
```
titanic_df.to_parquet('parquet_titanic_data.parquet', engine='pyarrow')
```

Save data to pickle

In [17]:
```
titanic_df.to_pickle('pickle_titanic_data.pkl')
```

Save data to SQLite

In [18]:
```
json_exchange_rate.to_sql('exchange_rate_sqlite_data', engine, chunksize=100)
```

3. Pandas Basic Data Inspection Functions

Show first 3 rows of data

In [19]:
```
titanic_df.head(3)
```

Out[19]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250

Show last 3 rows of data

In [20]:
```
titanic_df.tail(3)
```

Out[20]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
884	0	3	Miss. Catherine Helen Johnston	female	7.0	1	2	23.45
885	1	1	Mr. Karl Howell Behr	male	26.0	0	0	30.00
886	0	3	Mr. Patrick Dooley	male	32.0	0	0	7.75

Show number of rows and columns

In [21]:
```
titanic_df.shape
```

Out[21]: (887, 8)

Show data types, columns and memory

In [22]:
```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 887 entries, 0 to 886
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Survived                 887 non-null    int64  
 1   Pclass                   887 non-null    int64  
 2   Name                     887 non-null    object 
 3   Sex                      887 non-null    object 
 4   Age                      887 non-null    float64
 5   Siblings/Spouses Aboard  887 non-null    int64  
 6   Parents/Children Aboard  887 non-null    int64  
 7   Fare                     887 non-null    float64
dtypes: float64(2), int64(4), object(2)
memory usage: 55.6+ KB
```

Show data types

In [23]:
```
titanic_df.dtypes
```

Out[23]:
```
Survived                   int64
Pclass                     int64
Name                      object
Sex                       object
Age                      float64
Siblings/Spouses Aboard    int64
Parents/Children Aboard    int64
Fare                     float64
dtype: object
```

Show all columns

In [24]:
```
titanic_df.columns
```

Out[24]:
```
Index(['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'Siblings/Spouses Aboard',
       'Parents/Children Aboard', 'Fare'],
      dtype='object')
```

Show statistical summary

In [25]:
```
titanic_df.describe()
```

Out[25]:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
count	887.000000	887.000000	887.000000	887.000000	887.000000	887.00000
mean	0.385569	2.305524	29.471443	0.525366	0.383315	32.30542
std	0.487004	0.836662	14.121908	1.104669	0.807466	49.78204
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.00000
25%	0.000000	2.000000	20.250000	0.000000	0.000000	7.92500
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.45420
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.13750
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.32920

In [26]:
```
titanic_df.describe(include='all')
```

Out[26]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
count	887.000000	887.000000	887	887	887.000000	887.000000	887.000000	887.00000
unique	NaN	NaN	887	2	NaN	NaN	NaN	NaN
top	NaN	NaN	Mr. Gerious Youseff	male	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	573	NaN	NaN	NaN	NaN
mean	0.385569	2.305524	NaN	NaN	29.471443	0.525366	0.383315	32.30542
std	0.487004	0.836662	NaN	NaN	14.121908	1.104669	0.807466	49.78204
min	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	0.00000
25%	0.000000	2.000000	NaN	NaN	20.250000	0.000000	0.000000	7.92500
50%	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	14.45420
75%	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	31.13750
max	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	512.32920

Show unique values and there frequency

In [27]:
```
titanic_df['Sex'].value_counts()
```

Out[27]:
```
male      573
female    314
Name: Sex, dtype: int64
```

Return dimesnion of the data frame

In [28]:
```
titanic_df.ndim
```

Out[28]: 2

Show number of elemnts in the data frame

In [29]:
```
titanic_df.size
```

Out[29]: 7096

Rename columns

In [30]:
```
titanic_df.columns=['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'Siblings/Spouses Aboard','Parents/Children Aboard',
'Fare']
```

In [31]:
```
gdp_df.columns=['country','continent','imf_estimate','imf_year','un_estimate','un_year','wb_estimate','wb_year']
gdp_df.head()
```

Out[31]:

	country	continent	imf_estimate	imf_year	un_estimate	un_year	wb_estimate	wb_year
0	United States	Americas	22939580.0	2021	21433226	2019	20936600.0	2020
1	China	Asia	16862979.0	[n 2]2021	14342933	[n 3]2019	14722731.0	2020
2	Japan	Asia	5103110.0	2021	5082465	2019	4975415.0	2020
3	Germany	Europe	4230172.0	2021	3861123	2019	3806060.0	2020
4	United Kingdom	Europe	3108416.0	2021	2826441	2019	2707744.0	2020

Create new column

In [32]:
```
titanic_df['New_Fare']=titanic_df['Fare']+100
titanic_df.head()
```

Out[32]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500	107.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833	171.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250	107.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000	153.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500	108.0500

In [33]:
```
gdp_df['imf_estimate_log']=np.log(gdp_df['imf_estimate'])
gdp_df.head()
```

Out[33]:

	country	continent	imf_estimate	imf_year	un_estimate	un_year	wb_estimate	wb_year	imf_estimate_log
0	United States	Americas	22939580.0	2021	21433226	2019	20936600.0	2020	16.948374
1	China	Asia	16862979.0	[n 2]2021	14342933	[n 3]2019	14722731.0	2020	16.640631
2	Japan	Asia	5103110.0	2021	5082465	2019	4975415.0	2020	15.445361
3	Germany	Europe	4230172.0	2021	3861123	2019	3806060.0	2020	15.257753
4	United Kingdom	Europe	3108416.0	2021	2826441	2019	2707744.0	2020	14.949624

Check memory usage

In [34]:
```
titanic_df.memory_usage()
```

Out[34]:
```
Index                       128
Survived                   7096
Pclass                     7096
Name                       7096
Sex                        7096
Age                        7096
Siblings/Spouses Aboard    7096
Parents/Children Aboard    7096
Fare                       7096
New_Fare                   7096
dtype: int64
```

4. Pandas Data Types Converstions

In [35]:
```
datatypes_df = pd.DataFrame(
    {
        "Students": ["Tom", "Peter", "Mary", "Smith"],
        "Reg_No": [1790, 1731, 1780, 1755],
        "Reg_Date": ["15/01/2021", "16/01/2021", "19/01/2021", "27/01/2021"],
        "Math": ["79.00", "67.00", "84.00", "70.00"],
        "Physics": ["60", "70", "50", "90"],
        "Computer": ["65.80", "80", "70", "75"],
    }
)

datatypes_df
```

Out[35]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer
0	Tom	1790	15/01/2021	79.00	60	65.80
1	Peter	1731	16/01/2021	67.00	70	80
2	Mary	1780	19/01/2021	84.00	50	70
3	Smith	1755	27/01/2021	70.00	90	75

Check for data types

In [36]:
```
datatypes_df.dtypes
```

Out[36]:
```
Students    object
Reg_No       int64
Reg_Date    object
Math        object
Physics     object
Computer    object
dtype: object
```

Convert object to integer

Let's convert Physics column to interger

```
In [37]: datatypes_df['Physics']=datatypes_df['Physics'].astype(np.int)
```

```
In [38]: datatypes_df.dtypes
```

```
Out[38]: Students    object
         Reg_No       int64
         Reg_Date    object
         Math        object
         Physics      int32
         Computer    object
         dtype: object
```

Convert object to float

Let's convert Math and Computer columns to float

```
In [39]: datatypes_df[['Math','Computer']]=datatypes_df[['Math','Computer']].astype(np.float)
```

```
In [40]: datatypes_df.dtypes
```

```
Out[40]: Students     object
         Reg_No        int64
         Reg_Date     object
         Math        float64
         Physics       int32
         Computer    float64
         dtype: object
```

Convert object to Date

Let's convert Reg_Date column to valid pandas date

In [41]:
```
# datatypes_df['Reg_Date']=pd.to_datetime(datatypes_df['Reg_Date']) # You can also pass in format argument format='%Y/%m%d'
datatypes_df['Reg_Date']=pd.to_datetime(datatypes_df['Reg_Date'],format='%d/%m/%Y')
```

In [42]:
```
datatypes_df.dtypes
```

Out[42]:
```
Students            object
Reg_No               int64
Reg_Date    datetime64[ns]
Math               float64
Physics              int32
Computer           float64
dtype: object
```

Convert int to String/object

Let's convert Reg_No column to String/Object

In [43]:
```
datatypes_df['Reg_No']=datatypes_df['Reg_No'].astype(str)
```

In [44]:
```
datatypes_df.dtypes
```

Out[44]:
```
Students            object
Reg_No              object
Reg_Date    datetime64[ns]
Math               float64
Physics              int32
Computer           float64
dtype: object
```

Convert int to float to integer

Let's convert Math column to Integer

```
In [45]: datatypes_df['Math']=datatypes_df['Math'].astype(np.int)
```

```
In [46]: datatypes_df.dtypes
```

```
Out[46]: Students            object
         Reg_No              object
         Reg_Date    datetime64[ns]
         Math                 int32
         Physics              int32
         Computer           float64
         dtype: object
```

```
In [47]: datatypes_df
```

Out[47]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer
0	Tom	1790	2021-01-15	79	60	65.8
1	Peter	1731	2021-01-16	67	70	80.0
2	Mary	1780	2021-01-19	84	50	70.0
3	Smith	1755	2021-01-27	70	90	75.0

5. Select, Sort and Filter Data

Select specific columns and all records

In [48]:
```
# Select one column only
titanic_df['Survived']
# Use . syntax if the column name don't have spaces
titanic_df.Survived
# Select more than one column
titanic_df[['Survived','Pclass','Age']]
```

Out[48]:

	Survived	Pclass	Age
0	0	3	22.0
1	1	1	38.0
2	1	3	26.0
3	1	1	35.0
4	0	3	35.0
...	...	...	...
882	0	2	27.0
883	1	1	19.0
884	0	3	7.0
885	1	1	26.0
886	0	3	32.0

887 rows × 3 columns

In [49]:
```
titanic_df.head(10)
```

Out[49]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500	107.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833	171.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250	107.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000	153.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500	108.0500
5	0	3	Mr. James Moran	male	27.0	0	0	8.4583	108.4583
6	0	1	Mr. Timothy J McCarthy	male	54.0	0	0	51.8625	151.8625
7	0	3	Master. Gosta Leonard Palsson	male	2.0	3	1	21.0750	121.0750
8	1	3	Mrs. Oscar W (Elisabeth Vilhelmina Berg) Johnson	female	27.0	0	2	11.1333	111.1333
9	1	2	Mrs. Nicholas (Adele Achem) Nasser	female	14.0	1	0	30.0708	130.0708

Select data by position

In [50]:
```
titanic_df.iloc[:10] # select first 10 records
titanic_df.iloc[-10:] # select last 10 records
titanic_df.iloc[5:8] # select records from 5 to 8 records. Including 5 excluding 8
titanic_df.iloc[0:-10] # select all data excluding last 10 records
titanic_df.iloc[0:,0:2] # select first two columns with all records
titanic_df.iloc[0:,3:5] # select all rows and only columns from 3 to 5.
titanic_df.iloc[:5,0:-1] # select first 5 records and all columns excluding last 1 column
```

Out[50]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500

Sort data ascending

In [51]:
```
titanic_df.sort_values(by='Age').head()
```

Out[51]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
799	1	3	Master. Assad Alexander Thomas	male	0.42	0	1	8.5167	108.5167
751	1	2	Master. Viljo Hamalainen	male	0.67	1	1	14.5000	114.5000
641	1	3	Miss. Eugenie Baclini	female	0.75	2	1	19.2583	119.2583
466	1	3	Miss. Helene Barbara Baclini	female	0.75	2	1	19.2583	119.2583
827	1	2	Master. George Sibley Richards	male	0.83	1	1	18.7500	118.7500

Sort descending

In [52]:
```
titanic_df.sort_values(by='Age', ascending=False).head()
```

Out[52]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
627	1	1	Mr. Algernon Henry Wilson Barkworth	male	80.0	0	0	30.0000	130.0000
847	0	3	Mr. Johan Svensson	male	74.0	0	0	7.7750	107.7750
490	0	1	Mr. Ramon Artagaveytia	male	71.0	0	0	49.5042	149.5042
95	0	1	Mr. George B Goldschmidt	male	71.0	0	0	34.6542	134.6542
115	0	3	Mr. Patrick Connors	male	70.5	0	0	7.7500	107.7500

Sort data by two columns

In [53]:
```
titanic_df.sort_values(by=['Age','Fare'], ascending=[True,False]).head() # Sort age by ascending and then Fare by descending
```

Out[53]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
799	1	3	Master. Assad Alexander Thomas	male	0.42	0	1	8.5167	108.5167
751	1	2	Master. Viljo Hamalainen	male	0.67	1	1	14.5000	114.5000
466	1	3	Miss. Helene Barbara Baclini	female	0.75	2	1	19.2583	119.2583
641	1	3	Miss. Eugenie Baclini	female	0.75	2	1	19.2583	119.2583
77	1	2	Master. Alden Gates Caldwell	male	0.83	0	2	29.0000	129.0000

Filter data where age > 50

In [54]:
```
titanic_df[titanic_df['Age']>50].head()
```

Out[54]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
6	0	1	Mr. Timothy J McCarthy	male	54.0	0	0	51.8625	151.8625
11	1	1	Miss. Elizabeth Bonnell	female	58.0	0	0	26.5500	126.5500
15	1	2	Mrs. (Mary D Kingcome) Hewlett	female	55.0	0	0	16.0000	116.0000
33	0	2	Mr. Edward H Wheadon	male	66.0	0	0	10.5000	110.5000
53	0	1	Mr. Engelhart Cornelius Ostby	male	65.0	0	1	61.9792	161.9792

Filter data where age is between 30 and 40

```
In [55]: titanic_df[(titanic_df['Age']>30) & (titanic_df['Age']<40)].head()
```

Out[55]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833	171.2833
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000	153.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500	108.0500
13	0	3	Mr. Anders Johan Andersson	male	39.0	1	5	31.2750	131.2750
18	0	3	Mrs. Julius (Emelia Maria Vandemoortele) Vande...	female	31.0	1	0	18.0000	118.0000

Using IN function

In [56]:
```
titanic_df[titanic_df['Sex'].isin(['male'])].head()
```

Out[56]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500	107.2500
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500	108.0500
5	0	3	Mr. James Moran	male	27.0	0	0	8.4583	108.4583
6	0	1	Mr. Timothy J McCarthy	male	54.0	0	0	51.8625	151.8625
7	0	3	Master. Gosta Leonard Palsson	male	2.0	3	1	21.0750	121.0750

Using NOT IN

In [57]:
```
titanic_df[~titanic_df['Sex'].isin(['male'])].head()
```

Out[57]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833	171.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250	107.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000	153.1000
8	1	3	Mrs. Oscar W (Elisabeth Vilhelmina Berg) Johnson	female	27.0	0	2	11.1333	111.1333
9	1	2	Mrs. Nicholas (Adele Achem) Nasser	female	14.0	1	0	30.0708	130.0708

6. Reshaping DataFrames

In [58]:
```
gdp_df=pd.read_csv("gdp.csv")
gdp_df.head()
```

Out[58]:

	country	continent	imf_estimate	imf_year	un_estimate	un_year	wb_estimate	wb_year
0	United States	Americas	22939580.0	2021	21433226	2019	20936600.0	2020
1	China	Asia	16862979.0	[n 2]2021	14342933	[n 3]2019	14722731.0	2020
2	Japan	Asia	5103110.0	2021	5082465	2019	4975415.0	2020
3	Germany	Europe	4230172.0	2021	3861123	2019	3806060.0	2020
4	United Kingdom	Europe	3108416.0	2021	2826441	2019	2707744.0	2020

Tanspose Data

In [59]:
```
gdp_df.T
```

Out[59]:

	0	1	2	3	4	5	6	7	
country	United States	China	Japan	Germany	United Kingdom	India	France	Italy	Canada
continent	Americas	Asia	Asia	Europe	Europe	Asia	Europe	Europe	Americas
imf_estimate	2.29396e+07	1.6863e+07	5.10311e+06	4.23017e+06	3.10842e+06	2.94606e+06	2.94043e+06	2.12023e+06	2.01598e+0
imf_year	2021	[n 2]2021	2021	2021	2021	2021	2021	2021	2021
un_estimate	21433226	14342933	5082465	3861123	2826441	2891582	2715518	2003576	1741496
un_year	2019	[n 3]2019	2019	2019	2019	2019	2019	2019	2019
wb_estimate	2.09366e+07	1.47227e+07	4.97542e+06	3.80606e+06	2.70774e+06	2.62298e+06	2.603e+06	1.88644e+06	1.64341e+0
wb_year	2020	2020	2020	2020	2020	2020	2020	2020	2020

8 rows × 213 columns

Groupby

Groupby continents

In [60]:
```
groups=gdp_df.groupby(['continent'])
groups
```

Out[60]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002AB12D3FB70>
```

Iterate through groups

In [61]:
```
for i,j in groups:
    print(i)
#     print(j)
```

```
Africa
Americas
Asia
Europe
Oceania
```

Select groups

In [62]:
```
groups.get_group('Asia').head()
```

Out[62]:

	country	continent	imf_estimate	imf_year	un_estimate	un_year	wb_estimate	wb_year
1	China	Asia	16862979.0	[n 2]2021	14342933	[n 3]2019	14722731.0	2020
2	Japan	Asia	5103110.0	2021	5082465	2019	4975415.0	2020
5	India	Asia	2946061.0	2021	2891582	2019	2622984.0	2020
9	South Korea	Asia	1823852.0	2021	1646539	2019	1630525.0	2020
15	Indonesia	Asia	1150245.0	2021	1119190	2019	1058424.0	2020

Groupby with aggregation

In [63]:
```
# gdp_df.groupby(['continent']).aggregate(np.sum) # first approach
gdp_df.groupby(['continent']).sum() #second approach
```

Out[63]:

	imf_estimate	un_estimate	wb_estimate
continent			
Africa	2692597.0	2460570	2376597.0
Americas	30046495.0	28619051	27562046.0
Asia	36869524.0	33165597	32307365.0
Europe	23481224.0	21644976	20850775.0
Oceania	1894508.0	1638761	1582488.0

Groupby and passing multiple aggregation

In [64]:
```
gdp_df.groupby(['continent']).agg([np.sum,np.mean,np.std]).reset_index()
```

Out[64]:

	continent	imf_estimate			un_estimate			wb_estimate		
		sum	mean	std	sum	mean	std	sum	mean	
0	Africa	2692597.0	49862.907407	9.965247e+04	2460570	44737.636364	8.954446e+04	2376597.0	44011.055556	8.579816
1	Americas	30046495.0	834624.861111	3.817579e+06	28619051	622153.282609	3.162716e+06	27562046.0	640977.813953	3.189404
2	Asia	36869524.0	737390.480000	2.480266e+06	33165597	650305.823529	2.125590e+06	32307365.0	646147.300000	2.186434
3	Europe	23481224.0	559076.761905	9.436804e+05	21644976	491931.272727	8.593517e+05	20850775.0	473881.250000	8.269053
4	Oceania	1894508.0	135322.000000	4.296308e+05	1638761	96397.705882	3.345344e+05	1582488.0	98905.500000	3.327190

Groupby with aggregation and set index

In [65]:
```
# gdp_df.groupby(['continent'],as_index=False).sum() #first approach
gdp_df.groupby(['continent']).sum().reset_index() #second approach
```

Out[65]:

	continent	imf_estimate	un_estimate	wb_estimate
0	Africa	2692597.0	2460570	2376597.0
1	Americas	30046495.0	28619051	27562046.0
2	Asia	36869524.0	33165597	32307365.0
3	Europe	23481224.0	21644976	20850775.0
4	Oceania	1894508.0	1638761	1582488.0

Pivot

Reshape dataframe from long format to wide format. NOTE: Pivot does not deal with repeated/duplicated values in index. Instead we use Pivot Table

In [66]:
```
continental_temperature_df=pd.read_csv('continental_temperature.csv')
```

In [67]:
```
# Add Temperature in Farenheit
continental_temperature_df['AvgTemperature_FH']=continental_temperature_df['AvgTemperature']*(9/5)+32

print(continental_temperature_df.head().to_markdown(tablefmt='grid'))
```

```
+----+----------+--------+-----------------+--------------------+
|    | Region   |   Year |   AvgTemperature |   AvgTemperature_FH |
+====+==========+========+=================+====================+
|  0 | Africa   |   1995 |        0.976737 |            33.7581 |
+----+----------+--------+-----------------+--------------------+
|  1 | Africa   |   1996 |       48.3484   |           119.027  |
+----+----------+--------+-----------------+--------------------+
|  2 | Africa   |   1997 |       37.2825   |            99.1085 |
+----+----------+--------+-----------------+--------------------+
|  3 | Africa   |   1998 |       30.3271   |            86.5887 |
+----+----------+--------+-----------------+--------------------+
|  4 | Africa   |   1999 |       14.1539   |            57.477  |
+----+----------+--------+-----------------+--------------------+
```

Pivot data by Region

In [68]:
```
continental_temperature_df.pivot(index='Year',columns='Region',values=['AvgTemperature']).head()
```

Out[68]:

	AvgTemperature						
Region	**Africa**	**Asia**	**Australia/South Pacific**	**Europe**	**Middle East**	**North America**	**South/Central America & Carribean**
Year							
1995	0.976737	51.184689	61.126347	38.634474	55.416305	54.884315	36.252988
1996	48.348380	60.293226	60.811293	36.537590	57.235675	51.071101	53.665272
1997	37.282495	57.963918	61.577032	41.291805	58.599941	53.795868	55.117885
1998	30.327075	53.736634	49.237078	40.936384	58.497397	54.517819	49.526641
1999	14.153869	55.595147	61.653699	42.250435	59.681566	55.645530	58.425118

Pivot with more than one measure

```
In [69]: continental_temperature_df.pivot(index='Year',columns='Region',values=['AvgTemperature','AvgTemperature_FH']).head()
```

Out[69]:

	AvgTemperature							AvgTemperature_FH		
Region	**Africa**	**Asia**	**Australia/South Pacific**	**Europe**	**Middle East**	**North America**	**South/Central America & Carribean**	**Africa**	**Asia**	**Australia/Sou Pacific**
Year										
1995	0.976737	51.184689	61.126347	38.634474	55.416305	54.884315	36.252988	33.758127	124.132439	142.027425
1996	48.348380	60.293226	60.811293	36.537590	57.235675	51.071101	53.665272	119.027084	140.527806	141.460328
1997	37.282495	57.963918	61.577032	41.291805	58.599941	53.795868	55.117885	99.108491	136.335052	142.838658
1998	30.327075	53.736634	49.237078	40.936384	58.497397	54.517819	49.526641	86.588735	128.725941	120.626740
1999	14.153869	55.595147	61.653699	42.250435	59.681566	55.645530	58.425118	57.476964	132.071264	142.976658

Pivot Table

Create a spreadsheet-style pivot table as a DataFrame. For repeated values we specify the aggregation function

Pivot Table with one aggregation

In [70]:
```
pd.pivot_table(continental_temperature_df,index=['Year'],columns=['Region'],values='AvgTemperature',aggfunc=np.mean).head()
```

Out[70]:

Region	Africa	Asia	Australia/South Pacific	Europe	Middle East	North America	South/Central America & Carribean
Year							
1995	0.976737	51.184689	61.126347	38.634474	55.416305	54.884315	36.252988
1996	48.348380	60.293226	60.811293	36.537590	57.235675	51.071101	53.665272
1997	37.282495	57.963918	61.577032	41.291805	58.599941	53.795868	55.117885
1998	30.327075	53.736634	49.237078	40.936384	58.497397	54.517819	49.526641
1999	14.153869	55.595147	61.653699	42.250435	59.681566	55.645530	58.425118

Pivot Table with Totals

In [71]:
```
pd.pivot_table(continental_temperature_df,index=['Year'],columns=['Region'],values='AvgTemperature',
               aggfunc=np.mean,margins=True,margins_name='Mean')
```

Out[71]:

Region	Africa	Asia	Australia/South Pacific	Europe	Middle East	North America	South/Central America & Carribean	Mean
Year								
1995	0.976737	51.184689	61.126347	38.634474	55.416305	54.884315	36.252988	42.639408
1996	48.348380	60.293226	60.811293	36.537590	57.235675	51.071101	53.665272	52.566077
1997	37.282495	57.963918	61.577032	41.291805	58.599941	53.795868	55.117885	52.232706
1998	30.327075	53.736634	49.237078	40.936384	58.497397	54.517819	49.526641	48.111290
1999	14.153869	55.595147	61.653699	42.250435	59.681566	55.645530	58.425118	49.629338
2000	30.256454	54.118454	61.784699	46.421670	60.211397	55.084793	61.089126	52.709513
2001	39.146197	60.571804	61.309498	44.922228	72.698571	55.915848	64.430904	56.999293
2002	34.032093	59.585393	59.302055	43.831409	71.000646	53.367261	62.904055	54.860416
2003	34.280538	63.260892	60.405982	13.504275	73.234932	54.524121	60.304953	51.359385
2004	41.053458	62.384676	61.856512	47.620571	72.056284	55.114735	57.014710	56.728707
2005	47.777421	61.214857	62.342740	48.859400	71.696321	55.386966	68.041797	59.331357
2006	48.000331	162.959178	22.066667	48.090954	70.944814	56.112713	62.355933	67.218656
2007	64.601896	62.564065	62.232192	50.787129	19.231481	55.273890	63.332367	54.003289
2008	61.684503	64.241624	60.662750	51.060835	72.814019	54.032491	61.317346	60.830510
2009	68.413249	66.562027	62.203927	49.921038	72.982887	54.647723	67.069338	63.114313
2010	59.913363	67.625579	62.230913	47.098558	74.841066	55.666199	65.000600	61.768040
2011	61.297446	67.448425	62.133881	49.965203	71.509041	56.394212	68.361003	62.444173
2012	69.160824	67.073514	61.299863	49.115775	200.381547	57.924447	70.458905	82.202125
2013	69.381457	167.478655	62.669954	51.170174	74.096354	55.288881	113.894057	84.854219
2014	69.343831	67.266177	62.405388	52.338112	71.722107	55.040758	59.681527	62.542557
2015	70.118306	67.695330	61.630146	51.775807	72.165952	57.170802	72.058691	64.659291

Region	Africa	Asia	Australia/South Pacific	Europe	Middle East	North America	South/Central America & Carribean	Mean
Year								
2016	71.663865	67.075865	61.365027	50.392416	71.311349	58.057078	70.048035	64.273377
2017	71.761922	68.402059	62.892557	51.682540	70.180969	57.826738	129.952968	73.242822
2018	69.369124	66.411548	61.043333	50.207419	73.461201	56.860521	69.446720	63.828552
2019	70.283769	63.419344	60.729635	49.605295	71.476374	56.615323	69.635753	63.109356
2020	74.867798	61.685705	68.114303	44.800303	63.370647	17.406831	72.515579	57.537309
Mean	52.211400	70.300723	59.811057	45.877762	71.569956	53.985652	66.996241	60.107541

Pivot Table with multiple aggregations

In [72]:
```
pd.pivot_table(continental_temperature_df,index=['Year'],columns=['Region'],values=['AvgTemperature','AvgTemperature_FH'],
                   aggfunc={'AvgTemperature':np.mean,'AvgTemperature_FH':[np.min,np.max,np.mean,np.std]}).head()
```

Out[72]:

	AvgTemperature							AvgTemperature_FH		
	mean							amax		
Region	Africa	Asia	Australia/South Pacific	Europe	Middle East	North America	South/Central America & Carribean	Africa	Asia	Australia/Sou Pacific
Year										
1995	0.976737	51.184689	61.126347	38.634474	55.416305	54.884315	36.252988	33.758127	124.132439	142.027425
1996	48.348380	60.293226	60.811293	36.537590	57.235675	51.071101	53.665272	119.027084	140.527806	141.460328
1997	37.282495	57.963918	61.577032	41.291805	58.599941	53.795868	55.117885	99.108491	136.335052	142.838658
1998	30.327075	53.736634	49.237078	40.936384	58.497397	54.517819	49.526641	86.588735	128.725941	120.626740
1999	14.153869	55.595147	61.653699	42.250435	59.681566	55.645530	58.425118	57.476964	132.071264	142.976658

5 rows × 28 columns

In [73]:
```
pd.pivot_table(continental_temperature_df,index=['Year'],columns=['Region'],values=['AvgTemperature','AvgTemperature_F
H'],
                aggfunc={'AvgTemperature':np.mean,'AvgTemperature_FH':[np.min,np.max,np.mean,np.std]}).head().T
```

Out[73]:

		Year	1995	1996	1997	1998	1999
		Region					
AvgTemperature	mean	Africa	0.976737	48.348380	37.282495	30.327075	14.153869
		Asia	51.184689	60.293226	57.963918	53.736634	55.595147
		Australia/South Pacific	61.126347	60.811293	61.577032	49.237078	61.653699
		Europe	38.634474	36.537590	41.291805	40.936384	42.250435
		Middle East	55.416305	57.235675	58.599941	58.497397	59.681566
		North America	54.884315	51.071101	53.795868	54.517819	55.645530
		South/Central America & Carribean	36.252988	53.665272	55.117885	49.526641	58.425118
AvgTemperature_FH	amax	Africa	33.758127	119.027084	99.108491	86.588735	57.476964
		Asia	124.132439	140.527806	136.335052	128.725941	132.071264
		Australia/South Pacific	142.027425	141.460328	142.838658	120.626740	142.976658
		Europe	101.542053	97.767661	106.325249	105.685490	108.050784
		Middle East	131.749348	135.024215	137.479894	137.295315	139.426818
		North America	130.791768	123.927983	128.832563	130.132075	132.161954
		South/Central America & Carribean	97.255378	128.597489	131.212193	121.147954	137.165212
	amin	Africa	33.758127	119.027084	99.108491	86.588735	57.476964
		Asia	124.132439	140.527806	136.335052	128.725941	132.071264
		Australia/South Pacific	142.027425	141.460328	142.838658	120.626740	142.976658
		Europe	101.542053	97.767661	106.325249	105.685490	108.050784
		Middle East	131.749348	135.024215	137.479894	137.295315	139.426818
		North America	130.791768	123.927983	128.832563	130.132075	132.161954
		South/Central America & Carribean	97.255378	128.597489	131.212193	121.147954	137.165212
	mean	Africa	33.758127	119.027084	99.108491	86.588735	57.476964

		Year	1995	1996	1997	1998	1999
		Region					
		Asia	124.132439	140.527806	136.335052	128.725941	132.071264
		Australia/South Pacific	142.027425	141.460328	142.838658	120.626740	142.976658
		Europe	101.542053	97.767661	106.325249	105.685490	108.050784
		Middle East	131.749348	135.024215	137.479894	137.295315	139.426818
		North America	130.791768	123.927983	128.832563	130.132075	132.161954
		South/Central America & Carribean	97.255378	128.597489	131.212193	121.147954	137.165212

Stacking and Unstacking

In [74]:
```
# Let's first create a dataframe that we can stack
continental_temp_df=pd.pivot_table(continental_temperature_df,index=['Year'],columns=['Region'],
                                   values='AvgTemperature',aggfunc=np.mean)
continental_temp_df.head()
```

Out[74]:

Region	Africa	Asia	Australia/South Pacific	Europe	Middle East	North America	South/Central America & Carribean
Year							
1995	0.976737	51.184689	61.126347	38.634474	55.416305	54.884315	36.252988
1996	48.348380	60.293226	60.811293	36.537590	57.235675	51.071101	53.665272
1997	37.282495	57.963918	61.577032	41.291805	58.599941	53.795868	55.117885
1998	30.327075	53.736634	49.237078	40.936384	58.497397	54.517819	49.526641
1999	14.153869	55.595147	61.653699	42.250435	59.681566	55.645530	58.425118

Stack data by Year and Region

In [75]:
```
# Let's now stack the above dataframe with Year and Region
continental_temp_df.stack(0).reset_index().head()
```

Out[75]:

	Year	Region	0
0	1995	Africa	0.976737
1	1995	Asia	51.184689
2	1995	Australia/South Pacific	61.126347
3	1995	Europe	38.634474
4	1995	Middle East	55.416305

Unstack data by Region

In [76]:
```
stacked_df=gdp_df.groupby(['continent']).sum()
stacked_df.head()
```

Out[76]:

	imf_estimate	un_estimate	wb_estimate
continent			
Africa	2692597.0	2460570	2376597.0
Americas	30046495.0	28619051	27562046.0
Asia	36869524.0	33165597	32307365.0
Europe	23481224.0	21644976	20850775.0
Oceania	1894508.0	1638761	1582488.0

Melt

Unpivots a DataFrame from wide format to long format

In [77]:
```
continental_temp_df.reset_index().head()
```

Out[77]:

Region	Year	Africa	Asia	Australia/South Pacific	Europe	Middle East	North America	South/Central America & Carribean
0	1995	0.976737	51.184689	61.126347	38.634474	55.416305	54.884315	36.252988
1	1996	48.348380	60.293226	60.811293	36.537590	57.235675	51.071101	53.665272
2	1997	37.282495	57.963918	61.577032	41.291805	58.599941	53.795868	55.117885
3	1998	30.327075	53.736634	49.237078	40.936384	58.497397	54.517819	49.526641
4	1999	14.153869	55.595147	61.653699	42.250435	59.681566	55.645530	58.425118

In [78]:
```
year_region_temp_df=pd.melt(continental_temp_df.reset_index().head(),id_vars=['Year'],value_vars=['Asia','Africa','Eur
ope'],
        value_name='Temperature_C')
year_region_temp_df
```

Out[78]:

	Year	Region	Temperature_C
0	1995	Asia	51.184689
1	1996	Asia	60.293226
2	1997	Asia	57.963918
3	1998	Asia	53.736634
4	1999	Asia	55.595147
5	1995	Africa	0.976737
6	1996	Africa	48.348380
7	1997	Africa	37.282495
8	1998	Africa	30.327075
9	1999	Africa	14.153869
10	1995	Europe	38.634474
11	1996	Europe	36.537590
12	1997	Europe	41.291805
13	1998	Europe	40.936384
14	1999	Europe	42.250435

Cross-Tabulation

Pandas cross tabulation function allows us to summarize data similar to Pivot and Pivot_table.

In [79]:
```
# This is our original data before applying crosstab function
year_region_temp_df.head()
```

Out[79]:

	Year	Region	Temperature_C
0	1995	Asia	51.184689
1	1996	Asia	60.293226
2	1997	Asia	57.963918
3	1998	Asia	53.736634
4	1999	Asia	55.595147

Cross Tabulation on the Region and Year Columns

In [80]:
```
'''Let's apply crosstab function and round the results to 1 decimal place. Also note that we are using mean as
    our aggregation function '''
pd.crosstab(year_region_temp_df['Year'],year_region_temp_df['Region'],
            values=year_region_temp_df['Temperature_C'],aggfunc='mean').round(2)
```

Out[80]:

Region	Africa	Asia	Europe
Year			
1995	0.98	51.18	38.63
1996	48.35	60.29	36.54
1997	37.28	57.96	41.29
1998	30.33	53.74	40.94
1999	14.15	55.60	42.25

Adding Totals to crosstab table

In [81]:
```
pd.crosstab(year_region_temp_df['Year'],year_region_temp_df['Region'],values=year_region_temp_df['Temperature_C'],
            aggfunc='mean', margins=True, margins_name='Mean').round(2)
```

Out[81]:

Region	Africa	Asia	Europe	Mean
Year				
1995	0.98	51.18	38.63	30.27
1996	48.35	60.29	36.54	48.39
1997	37.28	57.96	41.29	45.51
1998	30.33	53.74	40.94	41.67
1999	14.15	55.60	42.25	37.33
Mean	26.22	55.75	39.93	40.63

Cross Tabulation with Normalization

In [82]:
```
titanic_df.head()
```

Out[82]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500	107.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833	171.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250	107.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000	153.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500	108.0500

In [83]:
```
'''Let's use titanic dataset and count the number of passenger who survived and the distribution of there gender.'''
pd.crosstab(titanic_df['Survived'],titanic_df['Sex'])
```

Out[83]:

Sex	female	male
Survived		
0	81	464
1	233	109

In [84]:
```
'''Now let's add total and see how it looks.'''
pd.crosstab(titanic_df['Survived'],titanic_df['Sex'], margins=True, margins_name='Total')
```

Out[84]:

Sex	female	male	Total
Survived			
0	81	464	545
1	233	109	342
Total	314	573	887

In [85]:
```
'''Let's normalize and multiply by 100%. Normalize argument computes the percent contribution of each value
    to the total. For example for female and 0 survived we take (81/887)*100 which we get 9.131905'''
pd.crosstab(titanic_df['Survived'],titanic_df['Sex'], margins=True, margins_name='Total',normalize=True)*100
```

Out[85]:

Sex	female	male	Total
Survived			
0	9.131905	52.311161	61.443067
1	26.268320	12.288613	38.556933
Total	35.400225	64.599775	100.000000

Crosstab and Normalize a cross rows

In [86]:
```
'''Let's normalize rowise by setting normalize='index'. For example to get the first value of female and 0 survived we take
    (81/545)*100 to get 14.862385'''
pd.crosstab(titanic_df['Survived'],titanic_df['Sex'], margins=True, margins_name='Total',normalize='index')*100
```

Out[86]:

Sex	female	male
Survived		
0	14.862385	85.137615
1	68.128655	31.871345
Total	35.400225	64.599775

Crosstab and Normalize a cross columns

In [87]:
```
'''Let's normalize rowise by setting normalize='columns'. For example to get the first value of female and 0 survived
 we take
    (81/314)*100 to get 25.79617834394904 '''
pd.crosstab(titanic_df['Survived'],titanic_df['Sex'], margins=True, margins_name='Total',normalize='columns')*100
```

Out[87]:

Sex	female	male	Total
Survived			
0	25.796178	80.977312	61.443067
1	74.203822	19.022688	38.556933

Add Grouping to our Crosstab function

In [88]:
```
titanic_df.head()
```

Out[88]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500	107.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833	171.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250	107.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000	153.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500	108.0500

In [89]:
```
'''Now let's add total ans see how it looks.'''
pd.crosstab([titanic_df['Survived'],titanic_df['Pclass']],[titanic_df['Sex'],titanic_df['Siblings/Spouses Aboard']],
            rownames=['Survival','Passenger Class'], colnames=['Gender','Siblings Aboard'], margins=True, margins_name
='Total')
```

Out[89]:

	Gender	female							male							Total
	Siblings Aboard	0	1	2	3	4	5	8	0	1	2	3	4	5	8	
Survival	Passenger Class															
0	**1**	1	2	0	0	0	0	0	59	16	1	1	0	0	0	80
	2	3	3	0	0	0	0	0	67	20	4	0	0	0	0	97
	3	33	21	3	7	4	1	3	231	35	7	4	11	4	4	368
1	**1**	48	38	3	2	0	0	0	29	15	1	0	0	0	0	136
	2	41	25	3	1	0	0	0	9	7	1	0	0	0	0	87
	3	48	17	4	1	2	0	0	35	10	1	0	1	0	0	119
Total		174	106	13	11	6	1	3	430	103	15	5	12	4	4	887

7. Missing Data

Create a sample data frame with missing values

In [90]:
```
missing_data_df = pd.DataFrame(
    {
        "Students": ["Tom", "Peter",np.nan, "Mary", "Tom","King","Tom","Mary",np.nan],
        "Exam_Date": ["15/01/2021", "16/01/2021", "19/01/2021", "27/01/2021", "16/01/2021",
                      "16/01/2021", "16/01/2021", "16/01/2021", "16/01/2021"],
        "Math": [79.00, 67.00,np.nan, 84.00, 70.00,np.nan,90.00,76.00,np.nan],
        "Physics":[63.00, np.nan, 60.00, np.nan,84.00, 77.00,55.00,np.nan,66.00],
        "Computer":[np.nan,78.00, 57.00, 88.00, 75.00,93.00,np.nan,70.00,np.nan],
    }
)

missing_data_df
```

Out[90]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	NaN
1	Peter	16/01/2021	67.0	NaN	78.0
2	NaN	19/01/2021	NaN	60.0	57.0
3	Mary	27/01/2021	84.0	NaN	88.0
4	Tom	16/01/2021	70.0	84.0	75.0
5	King	16/01/2021	NaN	77.0	93.0
6	Tom	16/01/2021	90.0	55.0	NaN
7	Mary	16/01/2021	76.0	NaN	70.0
8	NaN	16/01/2021	NaN	66.0	NaN

Show null values and return false if not null

```
In [91]: missing_data_df.isnull().head()
```

Out[91]:

	Students	Exam_Date	Math	Physics	Computer
0	False	False	False	False	True
1	False	False	False	True	False
2	True	False	True	False	False
3	False	False	False	True	False
4	False	False	False	False	False

Show count of null values

```
In [92]: missing_data_df.isnull().sum()
```

```
Out[92]: Students     2
         Exam_Date    0
         Math         3
         Physics      3
         Computer     3
         dtype: int64
```

Show not null values and return true if not null

In [93]:
```
missing_data_df.notnull().head()
```

Out[93]:

	Students	Exam_Date	Math	Physics	Computer
0	True	True	True	True	False
1	True	True	True	False	True
2	False	True	False	True	True
3	True	True	True	False	True
4	True	True	True	True	True

Drop entier row with all values null

In [94]:
```
missing_data_df.dropna(how='all')
```

Out[94]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	NaN
1	Peter	16/01/2021	67.0	NaN	78.0
2	NaN	19/01/2021	NaN	60.0	57.0
3	Mary	27/01/2021	84.0	NaN	88.0
4	Tom	16/01/2021	70.0	84.0	75.0
5	King	16/01/2021	NaN	77.0	93.0
6	Tom	16/01/2021	90.0	55.0	NaN
7	Mary	16/01/2021	76.0	NaN	70.0
8	NaN	16/01/2021	NaN	66.0	NaN

Drop column with any null value

```
In [95]: missing_data_df.dropna(how='any',axis=1)
```

Out[95]:

	Exam_Date
0	15/01/2021
1	16/01/2021
2	19/01/2021
3	27/01/2021
4	16/01/2021
5	16/01/2021
6	16/01/2021
7	16/01/2021
8	16/01/2021

Drop rows with any of specified columns have null

```
In [96]: missing_data_df.dropna(subset=['Math', 'Physics'], how='any')
```

Out[96]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	NaN
4	Tom	16/01/2021	70.0	84.0	75.0
6	Tom	16/01/2021	90.0	55.0	NaN

Drop rows with all of specified columns have null

```
In [97]: missing_data_df.dropna(subset=['Math', 'Physics'], how='all')
```

Out[97]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	NaN
1	Peter	16/01/2021	67.0	NaN	78.0
2	NaN	19/01/2021	NaN	60.0	57.0
3	Mary	27/01/2021	84.0	NaN	88.0
4	Tom	16/01/2021	70.0	84.0	75.0
5	King	16/01/2021	NaN	77.0	93.0
6	Tom	16/01/2021	90.0	55.0	NaN
7	Mary	16/01/2021	76.0	NaN	70.0
8	NaN	16/01/2021	NaN	66.0	NaN

Drop row with a given number of null values

In [98]:
```
missing_data_df.dropna(axis=1,thresh=2)
```

Out[98]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	NaN
1	Peter	16/01/2021	67.0	NaN	78.0
2	NaN	19/01/2021	NaN	60.0	57.0
3	Mary	27/01/2021	84.0	NaN	88.0
4	Tom	16/01/2021	70.0	84.0	75.0
5	King	16/01/2021	NaN	77.0	93.0
6	Tom	16/01/2021	90.0	55.0	NaN
7	Mary	16/01/2021	76.0	NaN	70.0
8	NaN	16/01/2021	NaN	66.0	NaN

Replace null values with a scalar value

In [99]:
```
missing_data_df.fillna(-999) # replace null values with -999
```

Out[99]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	-999.0
1	Peter	16/01/2021	67.0	-999.0	78.0
2	-999	19/01/2021	-999.0	60.0	57.0
3	Mary	27/01/2021	84.0	-999.0	88.0
4	Tom	16/01/2021	70.0	84.0	75.0
5	King	16/01/2021	-999.0	77.0	93.0
6	Tom	16/01/2021	90.0	55.0	-999.0
7	Mary	16/01/2021	76.0	-999.0	70.0
8	-999	16/01/2021	-999.0	66.0	-999.0

Backward Fill

In [100]:
```
missing_data_df.fillna(method='bfill')
```

Out[100]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	78.0
1	Peter	16/01/2021	67.0	60.0	78.0
2	Mary	19/01/2021	84.0	60.0	57.0
3	Mary	27/01/2021	84.0	84.0	88.0
4	Tom	16/01/2021	70.0	84.0	75.0
5	King	16/01/2021	90.0	77.0	93.0
6	Tom	16/01/2021	90.0	55.0	70.0
7	Mary	16/01/2021	76.0	66.0	70.0
8	NaN	16/01/2021	NaN	66.0	NaN

Forward Fill

In [101]:
```
missing_data_df.fillna(method='ffill')
```

Out[101]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	NaN
1	Peter	16/01/2021	67.0	63.0	78.0
2	Peter	19/01/2021	67.0	60.0	57.0
3	Mary	27/01/2021	84.0	60.0	88.0
4	Tom	16/01/2021	70.0	84.0	75.0
5	King	16/01/2021	70.0	77.0	93.0
6	Tom	16/01/2021	90.0	55.0	93.0
7	Mary	16/01/2021	76.0	55.0	70.0
8	Mary	16/01/2021	76.0	66.0	70.0

Replace missing value with a specific value

In [102]:
```
missing_data_df.replace(np.nan, 0)
```

Out[102]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	0.0
1	Peter	16/01/2021	67.0	0.0	78.0
2	0	19/01/2021	0.0	60.0	57.0
3	Mary	27/01/2021	84.0	0.0	88.0
4	Tom	16/01/2021	70.0	84.0	75.0
5	King	16/01/2021	0.0	77.0	93.0
6	Tom	16/01/2021	90.0	55.0	0.0
7	Mary	16/01/2021	76.0	0.0	70.0
8	0	16/01/2021	0.0	66.0	0.0

Impute null value with statistical measures

In [103]:
```
missing_data_df.fillna(missing_data_df.Math.mean()) # fillna null value in Math column with mean of the Math
missing_data_df.fillna(missing_data_df.Students.mode()) # fillna null value in Students column with mode of the Studen
ts
missing_data_df.fillna(missing_data_df.Computer.median()) # fillna null value in Computer column with median of the Co
mputer
```

Out[103]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	76.5
1	Peter	16/01/2021	67.0	76.5	78.0
2	76.5	19/01/2021	76.5	60.0	57.0
3	Mary	27/01/2021	84.0	76.5	88.0
4	Tom	16/01/2021	70.0	84.0	75.0
5	King	16/01/2021	76.5	77.0	93.0
6	Tom	16/01/2021	90.0	55.0	76.5
7	Mary	16/01/2021	76.0	76.5	70.0
8	76.5	16/01/2021	76.5	66.0	76.5

Interpolate missing values

In [104]:
```
missing_data_df.interpolate(method='linear')
```

Out[104]:

	Students	Exam_Date	Math	Physics	Computer
0	Tom	15/01/2021	79.0	63.0	NaN
1	Peter	16/01/2021	67.0	61.5	78.0
2	NaN	19/01/2021	75.5	60.0	57.0
3	Mary	27/01/2021	84.0	72.0	88.0
4	Tom	16/01/2021	70.0	84.0	75.0
5	King	16/01/2021	80.0	77.0	93.0
6	Tom	16/01/2021	90.0	55.0	81.5
7	Mary	16/01/2021	76.0	60.5	70.0
8	NaN	16/01/2021	76.0	66.0	70.0

8. Date and Time

Pnadas to_datetime function

In [105]:
```
clv_df=pd.read_csv('clv.csv')
clv_df['text_with_date']='Some text '+clv_df['InvoiceDate'] +'Other text'
clv_df.head()
```

Out[105]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	text_with_date
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01/12/2010	2.55	17850.0	United Kingdom	Some text 01/12/2010Other text
1	536365	71053	WHITE METAL LANTERN	6	01/12/2010	3.39	17850.0	United Kingdom	Some text 01/12/2010Other text
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01/12/2010	2.75	17850.0	United Kingdom	Some text 01/12/2010Other text
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01/12/2010	3.39	17850.0	United Kingdom	Some text 01/12/2010Other text
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01/12/2010	3.39	17850.0	United Kingdom	Some text 01/12/2010Other text

Check if the date columns has datetime type

In [106]:
```
clv_df.dtypes
```

Out[106]:
```
InvoiceNo          object
StockCode          object
Description        object
Quantity            int64
InvoiceDate        object
UnitPrice         float64
CustomerID        float64
Country            object
text_with_date     object
dtype: object
```

Convert a String Columns to Datetime object

In [107]:
```
# clv_df['valid_invoice_date_object']=pd.to_datetime(clv_df['InvoiceDate'])
clv_df['valid_invoice_date_object']=pd.to_datetime(clv_df['InvoiceDate'],format='%d/%m/%Y') # add format
clv_df['date_only']=pd.to_datetime(clv_df['text_with_date'],format='Some text %d/%m/%YOther text') # add format
clv_df.head()
```

Out[107]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	text_with_date	valid_invoice_date_ob
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01/12/2010	2.55	17850.0	United Kingdom	Some text 01/12/2010Other text	2010-12-01
1	536365	71053	WHITE METAL LANTERN	6	01/12/2010	3.39	17850.0	United Kingdom	Some text 01/12/2010Other text	2010-12-01
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01/12/2010	2.75	17850.0	United Kingdom	Some text 01/12/2010Other text	2010-12-01
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01/12/2010	3.39	17850.0	United Kingdom	Some text 01/12/2010Other text	2010-12-01
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01/12/2010	3.39	17850.0	United Kingdom	Some text 01/12/2010Other text	2010-12-01

In [108]:
```
clv_df.dtypes
```

Out[108]:
```
InvoiceNo                          object
StockCode                          object
Description                        object
Quantity                            int64
InvoiceDate                        object
UnitPrice                         float64
CustomerID                        float64
Country                            object
text_with_date                     object
valid_invoice_date_object    datetime64[ns]
date_only                    datetime64[ns]
dtype: object
```

Extract Year from date

In [109]:
```
clv_df['valid_invoice_date_object'].dt.year.head()
```

Out[109]:
```
0    2010
1    2010
2    2010
3    2010
4    2010
Name: valid_invoice_date_object, dtype: int64
```

Extract Month from date

In [110]:
```
clv_df['valid_invoice_date_object'].dt.month.head()
```

Out[110]:
```
0    12
1    12
2    12
3    12
4    12
Name: valid_invoice_date_object, dtype: int64
```

Extract Day from date

```
In [111]: clv_df['valid_invoice_date_object'].dt.day.head()
```

```
Out[111]: 0    1
          1    1
          2    1
          3    1
          4    1
          Name: valid_invoice_date_object, dtype: int64
```

Extract Day of Year from date

```
In [112]: clv_df['valid_invoice_date_object'].dt.dayofyear.head()
```

```
Out[112]: 0    335
          1    335
          2    335
          3    335
          4    335
          Name: valid_invoice_date_object, dtype: int64
```

Extract Week of Year from date

```
In [113]: clv_df['valid_invoice_date_object'].dt.isocalendar().week.head()
```

```
Out[113]: 0    48
          1    48
          2    48
          3    48
          4    48
          Name: week, dtype: UInt32
```

Extract Day of Year from date

In [114]:
```
# clv_df['valid_invoice_date_object'].dt.dayofweek.head()
clv_df['valid_invoice_date_object'].dt.isocalendar().day.head()
```

Out[114]:
```
0    3
1    3
2    3
3    3
4    3
Name: day, dtype: UInt32
```

Name of Day

In [115]:
```
def week_day(x):
    return pd.Timestamp(x).day_name()
```

In [116]:
```
clv_df['valid_invoice_date_object'].apply(lambda x: week_day(x)).head()
```

Out[116]:
```
0    Wednesday
1    Wednesday
2    Wednesday
3    Wednesday
4    Wednesday
Name: valid_invoice_date_object, dtype: object
```

Extract Quarter of Year from date

In [117]:
```
clv_df['valid_invoice_date_object'].dt.quarter.head()
```

Out[117]:
```
0    4
1    4
2    4
3    4
4    4
Name: valid_invoice_date_object, dtype: int64
```

Extract Number of Days in a month

In [118]:
```
clv_df['valid_invoice_date_object'].dt.days_in_month.head()
```

Out[118]:
```
0    31
1    31
2    31
3    31
4    31
Name: valid_invoice_date_object, dtype: int64
```

Check for leap year

In [119]:
```
clv_df['valid_invoice_date_object'].dt.is_leap_year.head()
```

Out[119]:
```
0    False
1    False
2    False
3    False
4    False
Name: valid_invoice_date_object, dtype: bool
```

Extract Hour from date

In [120]:
```
dates=pd.date_range('1/12/2022', periods = 100, freq ='H')
date_df=pd.DataFrame(dates,columns=['DatetimeColumn'])
date_df.head()
```

Out[120]:

	DatetimeColumn
0	2022-01-12 00:00:00
1	2022-01-12 01:00:00
2	2022-01-12 02:00:00
3	2022-01-12 03:00:00
4	2022-01-12 04:00:00

In [121]:
```
date_df['DatetimeColumn'].dt.hour.head()
```

Out[121]:
```
0    0
1    1
2    2
3    3
4    4
Name: DatetimeColumn, dtype: int64
```

Extract Minute from date

In [122]:
```
date_df['DatetimeColumn'].dt.minute.head()
```

Out[122]:
```
0    0
1    0
2    0
3    0
4    0
Name: DatetimeColumn, dtype: int64
```

Extract Second from date

In [123]:
```
date_df['DatetimeColumn'].dt.second.head()
```

Out[123]:
```
0    0
1    0
2    0
3    0
4    0
Name: DatetimeColumn, dtype: int64
```

Python Date Functions

Python date

In [124]:
```
from datetime import date
```

In [125]:
```
python_date=date.today()
print(python_date)
python_date.day
python_date.month
python_date.year
```

```
2022-01-13
```

Out[125]:
```
2022
```

Python time

In [126]:
```
from datetime import time
```

In [127]:
```
python_time=time(15,12,23,56)
print(python_time)
python_time.hour
python_time.minute
python_time.second
```

```
15:12:23.000056
```

Out[127]: 23

Python datetime

In [128]:
```
from datetime import datetime
```

In [129]:
```
python_datetime=datetime.now()
print(python_datetime)
python_datetime.date()
python_datetime.time()
python_datetime.strftime('%A') # Add formart to extract more features. Refer to Python documentation
# for more details https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes
```

```
2022-01-13 22:32:33.834339
```

Out[129]: 'Thursday'

Python Calendar

In [130]:
```
import calendar
```

Full Year calendar

In [131]:
```
print(calendar.calendar(2022))
```

```
                                  2022

      January               February              March
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                1  2      1  2  3  4  5  6      1  2  3  4  5  6
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   7  8  9 10 11 12 13
10 11 12 13 14 15 16  14 15 16 17 18 19 20  14 15 16 17 18 19 20
17 18 19 20 21 22 23  21 22 23 24 25 26 27  21 22 23 24 25 26 27
24 25 26 27 28 29 30  28                    28 29 30 31
31

       April                 May                  June
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
             1  2  3                  1            1  2  3  4  5
 4  5  6  7  8  9 10   2  3  4  5  6  7  8   6  7  8  9 10 11 12
11 12 13 14 15 16 17   9 10 11 12 13 14 15  13 14 15 16 17 18 19
18 19 20 21 22 23 24  16 17 18 19 20 21 22  20 21 22 23 24 25 26
25 26 27 28 29 30     23 24 25 26 27 28 29  27 28 29 30
                      30 31

        July                August              September
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
             1  2  3   1  2  3  4  5  6  7               1  2  3  4
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   5  6  7  8  9 10 11
11 12 13 14 15 16 17  15 16 17 18 19 20 21  12 13 14 15 16 17 18
18 19 20 21 22 23 24  22 23 24 25 26 27 28  19 20 21 22 23 24 25
25 26 27 28 29 30 31  29 30 31              26 27 28 29 30

      October              November             December
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                1  2      1  2  3  4  5  6            1  2  3  4
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   5  6  7  8  9 10 11
10 11 12 13 14 15 16  14 15 16 17 18 19 20  12 13 14 15 16 17 18
17 18 19 20 21 22 23  21 22 23 24 25 26 27  19 20 21 22 23 24 25
24 25 26 27 28 29 30  28 29 30              26 27 28 29 30 31
31
```

One Month Calendar

```
In [132]: print(calendar.month(2022,5))
```

```
      May 2022
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

9. Combining Data in Pandas

1. Concatenating Data Frames

This involves joining data frames along a specified axis (rowise or columnwise).

In [133]:
```
df1 = pd.DataFrame(
    {
        "A": ["A0", "A1", "A2", "A3"],
        "B": ["B0", "B1", "B2", "B3"],
        "C": ["C0", "C1", "C2", "C3"],
        "D": ["D0", "D1", "D2", "D3"],
    }
)

df2 = pd.DataFrame(
    {
        "A": ["A4", "A5", "A6", "A7"],
        "B": ["B4", "B5", "B6", "B7"],
        "C": ["C4", "C5", "C6", "C7"],
        "D": ["D4", "D5", "D6", "D7"],
    },
)

df3 = pd.DataFrame(
    {
        "A": ["A8", "A9", "A10", "A11"],
        "B": ["B8", "B9", "B10", "B11"],
        "C": ["C8", "C9", "C10", "C11"],
        "D": ["D8", "D9", "D10", "D11"],
    },
)

df4 = pd.DataFrame(
    {
        "B": ["B2", "B3", "B6", "B7"],
        "D": ["D2", "D3", "D6", "D7"],
        "F": ["F2", "F3", "F6", "F7"],
    },
)
```

In [134]:
```
df1
```

Out[134]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [135]:
```
df2
```

Out[135]:

	A	B	C	D
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

In [136]:
```
df3
```

Out[136]:

	A	B	C	D
0	A8	B8	C8	D8
1	A9	B9	C9	D9
2	A10	B10	C10	D10
3	A11	B11	C11	D11

In [137]:
```
df4
```

Out[137]:

	B	D	F
0	B2	D2	F2
1	B3	D3	F3
2	B6	D6	F6
3	B7	D7	F7

Stack dataframes on top of each other (rowise/axis=0)

We set ignore_index=True to reset the indexes of two dataframes to a unified index

In [138]:
```
# pd.concat([df1,df2], ignore_index=True) # Default concat function stacks dataframes on axis=0 (rowise)
pd.concat([df1,df2,df3], ignore_index=True,axis=0)
```

Out[138]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Pandas concat and set keys

In [139]:
```
pd.concat([df1,df2,df3],axis=0,keys=['df1','df2','df3'])
```

Out[139]:

		A	B	C	D
df1	**0**	A0	B0	C0	D0
	1	A1	B1	C1	D1
	2	A2	B2	C2	D2
	3	A3	B3	C3	D3
df2	**0**	A4	B4	C4	D4
	1	A5	B5	C5	D5
	2	A6	B6	C6	D6
	3	A7	B7	C7	D7
df3	**0**	A8	B8	C8	D8
	1	A9	B9	C9	D9
	2	A10	B10	C10	D10
	3	A11	B11	C11	D11

Stack dataframes next to each other (columnwise/axis=1)

In [140]:
```
pd.concat([df1,df4],axis=1)
```

Out[140]:

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	B2	D2	F2
1	A1	B1	C1	D1	B3	D3	F3
2	A2	B2	C2	D2	B6	D6	F6
3	A3	B3	C3	D3	B7	D7	F7

Stack dataframes next to each other (columnwise/axis=1) with keys

In [141]:
```
pd.concat([df1,df4],axis=1,keys=['df1','df4'])
```

Out[141]:

	df1				df4		
	A	**B**	**C**	**D**	**B**	**D**	**F**
0	A0	B0	C0	D0	B2	D2	F2
1	A1	B1	C1	D1	B3	D3	F3
2	A2	B2	C2	D2	B6	D6	F6
3	A3	B3	C3	D3	B7	D7	F7

Pandas append function

In [142]:
```
df1.append([df2,df3],ignore_index=True)
```

Out[142]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Pandas merge function

pandas has full-featured, high performance in-memory join operations idiomatically very similar to relational databases like SQL. These methods perform significantly better (in some cases well over an order of magnitude better) than other open source implementations (like base::merge.data.frame in R). The reason for this is careful algorithmic design and the internal layout of the data in DataFrame. Pandas Merge function provides an SQL-Like capability for joining two or more datasets.

In [143]:
```
hr_departments_df=pd.read_csv('hr_departments.csv')
hr_employees_df=pd.read_csv('hr_employees.csv')
```

In [144]:
```
hr_departments_df.head()
```

Out[144]:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
0	10	Administration	200.0	1700
1	20	Marketing	201.0	1800
2	30	Purchasing	114.0	1700
3	40	Human Resources	203.0	2400
4	50	Shipping	121.0	1500

In [145]:
```
hr_employees_df.head()
```

Out[145]:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
0	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	NaN
1	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	NaN
2	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	NaN
3	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	NaN
4	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	NaN

Pandas merge function with keys

Let's join Departments and Employee data with DEPARTMENT_ID as a common Column

In [146]:
```
pd.merge(hr_employees_df,hr_departments_df,on=['DEPARTMENT_ID']).head()
```

Out[146]:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
0	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	NaN
1	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	NaN
2	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	NaN
3	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	NaN
4	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	NaN

Pandas merge function with keys and inner join criteria

Inner join is equivalent to SQL INNER JOIN. Results in intersection of data with only matching keys from both dataframes

In [147]:
```
left = pd.DataFrame(
    {
        "key1": ["K0", "K0", "K1", "K2"],
        "key2": ["K0", "K1", "K0", "K1"],
        "A": ["A0", "A1", "A2", "A3"],
        "B": ["B0", "B1", "B2", "B3"],
    }
)

right = pd.DataFrame(
    {
        "key1": ["K0", "K1", "K1", "K2"],
        "key2": ["K0", "K0", "K0", "K0"],
        "C": ["C0", "C1", "C2", "C3"],
        "D": ["D0", "D1", "D2", "D3"],
    }
)
```

In [148]:
```
left
```

Out[148]:

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

In [149]:
```
right
```

Out[149]:

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Merge on left join

Left join resembles SQL LEFT OUTER JOIN where the resulting dataframe contains all the keys from left dataframe

```
In [150]: pd.merge(left,right,on=['key1','key2'],how='left')
```

Out[150]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN

Merge on right join

Left join resembles SQL RIGHT OUTER JOIN where the resulting dataframe contains all the keys from right dataframe

```
In [151]: pd.merge(left,right,on=['key1','key2'],how='right')
```

Out[151]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2
3	K2	K0	NaN	NaN	C3	D3

Merge on inner join

Inner join resembles SQL INNER JOIN resulting to intersection of keys from both dataframes

In [152]:
```
pd.merge(left,right,on=['key1','key2'],how='inner')
```

Out[152]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

Remove duplicates during joining

Removes duplicates to specified dataframe before joining

In [153]:
```
pd.merge(left.drop_duplicates(),right,on=['key1','key2'],how='inner')
```

Out[153]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

Validate join

We can validate our join criteria based on; one_to_one, one_to_many and many_to_many. If the condition is not satisfied it throws an error

```
In [154]: pd.merge(left.drop_duplicates(),right,on=['key1','key2'],how='inner',validate='many_to_many')
```

Out[154]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

Merge with outer join criteria

Outer join resembles SQL like FULL OUTER JOIN. Results in a union

```
In [155]: pd.merge(left,right,on=['key1','key2'],how='outer')
```

Out[155]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN
5	K2	K0	NaN	NaN	C3	D3

Merge with outer join criteria and indicator

The argument indicator=True creates a new column that describes the kind of join relationship in each records

In [156]:
```
pd.merge(left,right,on=['key1','key2'],how='outer',indicator=True)
```

Out[156]:

	key1	key2	A	B	C	D	_merge
0	K0	K0	A0	B0	C0	D0	both
1	K0	K1	A1	B1	NaN	NaN	left_only
2	K1	K0	A2	B2	C1	D1	both
3	K1	K0	A2	B2	C2	D2	both
4	K2	K1	A3	B3	NaN	NaN	left_only
5	K2	K0	NaN	NaN	C3	D3	right_only

Merge on cross join

This is synonymous to CROSS JOIN in SQL. It results in a cartesian product of rows in both dataframes. The cross argument works in Pandas version 1.2+. First we will check the pandas version. It's less than 1.2 we will upgrade it.

In [157]:
```
# Check pandas version
pd.__version__
```

Out[157]:
```
'1.1.5'
```

10. Pandas Apply Function

Create a Data Frame

In [158]:
```
apply_data_df = pd.DataFrame(
    {
        "Students": ["Tom", "Peter","Simon", "Mary", "Jane","King","Hillary","Ethan","Page"],
        "Math": [79.00, 67.00,80.00, 84.00, 70.00,60.00,90.00,76.00,75],
        "Physics":[63.00, 98, 60.00, 90,84.00, 77.00,55.00,70,66.00],
        "Computer":[84.00,78.00, 57.00, 88.00, 75.00,93.00,92.00,98.00,90.00],
    }
)

apply_data_df
```

Out[158]:

	Students	Math	Physics	Computer
0	Tom	79.0	63.0	84.0
1	Peter	67.0	98.0	78.0
2	Simon	80.0	60.0	57.0
3	Mary	84.0	90.0	88.0
4	Jane	70.0	84.0	75.0
5	King	60.0	77.0	93.0
6	Hillary	90.0	55.0	92.0
7	Ethan	76.0	70.0	98.0
8	Page	75.0	66.0	90.0

Get total for each Student

In [159]:
```
apply_data_df['Total']=apply_data_df[['Math','Physics','Computer']].apply(np.sum,axis=1)
apply_data_df
```

Out[159]:

	Students	Math	Physics	Computer	Total
0	Tom	79.0	63.0	84.0	226.0
1	Peter	67.0	98.0	78.0	243.0
2	Simon	80.0	60.0	57.0	197.0
3	Mary	84.0	90.0	88.0	262.0
4	Jane	70.0	84.0	75.0	229.0
5	King	60.0	77.0	93.0	230.0
6	Hillary	90.0	55.0	92.0	237.0
7	Ethan	76.0	70.0	98.0	244.0
8	Page	75.0	66.0	90.0	231.0

Get total for each Course

In [160]:
```
apply_data_df[['Math','Physics','Computer']].apply(np.sum,axis=0)
```

Out[160]:
```
Math        681.0
Physics     663.0
Computer    755.0
dtype: float64
```

Using apply function to convert data from float to int

In [161]:
```
apply_data_df[['Math','Physics','Computer']].apply(np.int64)
```

Out[161]:

	Math	Physics	Computer
0	79	63	84
1	67	98	78
2	80	60	57
3	84	90	88
4	70	84	75
5	60	77	93
6	90	55	92
7	76	70	98
8	75	66	90

Lambda function

In [162]:
```
add=lambda x,y:x+y
add(10,20)
```

Out[162]: 30

Using lambda function to add 5 to the Math course

In [163]:
```
apply_data_df['Math']=apply_data_df['Math'].apply(lambda x: x+5)
apply_data_df
```

Out[163]:

	Students	Math	Physics	Computer	Total
0	Tom	84.0	63.0	84.0	226.0
1	Peter	72.0	98.0	78.0	243.0
2	Simon	85.0	60.0	57.0	197.0
3	Mary	89.0	90.0	88.0	262.0
4	Jane	75.0	84.0	75.0	229.0
5	King	65.0	77.0	93.0	230.0
6	Hillary	95.0	55.0	92.0	237.0
7	Ethan	81.0	70.0	98.0	244.0
8	Page	80.0	66.0	90.0	231.0

Get Upper case names of students with lambda and apply

In [164]:
```
apply_data_df['Students'].apply(lambda x: x.upper())
```

Out[164]:
```
0        TOM
1      PETER
2      SIMON
3       MARY
4       JANE
5       KING
6    HILLARY
7      ETHAN
8       PAGE
Name: Students, dtype: object
```

Lambda with user defined function in apply. Similar to above solution

In [165]:
```
def upper_case(x):
    return x.upper()
```

In [166]:
```
apply_data_df['Students__Upper_Case_Names']=apply_data_df['Students'].apply(lambda x: upper_case(x))
apply_data_df
```

Out[166]:

	Students	Math	Physics	Computer	Total	Students__Upper_Case_Names
0	Tom	84.0	63.0	84.0	226.0	TOM
1	Peter	72.0	98.0	78.0	243.0	PETER
2	Simon	85.0	60.0	57.0	197.0	SIMON
3	Mary	89.0	90.0	88.0	262.0	MARY
4	Jane	75.0	84.0	75.0	229.0	JANE
5	King	65.0	77.0	93.0	230.0	KING
6	Hillary	95.0	55.0	92.0	237.0	HILLARY
7	Ethan	81.0	70.0	98.0	244.0	ETHAN
8	Page	80.0	66.0	90.0	231.0	PAGE

Use if-else with lambda and apply function

In [167]:
```
apply_data_df['Major_Grades']=apply_data_df['Total'].apply(lambda x: 'A' if x>250 else 'B')
apply_data_df
```

Out[167]:

	Students	Math	Physics	Computer	Total	Students__Upper_Case_Names	Major_Grades
0	Tom	84.0	63.0	84.0	226.0	TOM	B
1	Peter	72.0	98.0	78.0	243.0	PETER	B
2	Simon	85.0	60.0	57.0	197.0	SIMON	B
3	Mary	89.0	90.0	88.0	262.0	MARY	A
4	Jane	75.0	84.0	75.0	229.0	JANE	B
5	King	65.0	77.0	93.0	230.0	KING	B
6	Hillary	95.0	55.0	92.0	237.0	HILLARY	B
7	Ethan	81.0	70.0	98.0	244.0	ETHAN	B
8	Page	80.0	66.0	90.0	231.0	PAGE	B

Use if-elif-else with lambda and apply function

In [168]:
```
# We can make it as complex as we want
apply_data_df['Detailed_Grades']=apply_data_df['Total'].apply(lambda x: 'A' if x>250
                                                              else ('B' if (x>240) & (x<250)
                                                                    else ('C' if(x<240) & (x>200)
                                                                          else 'D') ))
apply_data_df
```

Out[168]:

	Students	Math	Physics	Computer	Total	Students__Upper_Case_Names	Major_Grades	Detailed_Grades
0	Tom	84.0	63.0	84.0	226.0	TOM	B	C
1	Peter	72.0	98.0	78.0	243.0	PETER	B	B
2	Simon	85.0	60.0	57.0	197.0	SIMON	B	D
3	Mary	89.0	90.0	88.0	262.0	MARY	A	A
4	Jane	75.0	84.0	75.0	229.0	JANE	B	C
5	King	65.0	77.0	93.0	230.0	KING	B	C
6	Hillary	95.0	55.0	92.0	237.0	HILLARY	B	C
7	Ethan	81.0	70.0	98.0	244.0	ETHAN	B	B
8	Page	80.0	66.0	90.0	231.0	PAGE	B	C

Using map function

In [169]:
```
apply_data_df['Positions']=apply_data_df['Detailed_Grades'].map({'A':1,'B':2,'C':3,'D':4})
apply_data_df
```

Out[169]:

	Students	Math	Physics	Computer	Total	Students__Upper_Case_Names	Major_Grades	Detailed_Grades	Positions
0	Tom	84.0	63.0	84.0	226.0	TOM	B	C	3
1	Peter	72.0	98.0	78.0	243.0	PETER	B	B	2
2	Simon	85.0	60.0	57.0	197.0	SIMON	B	D	4
3	Mary	89.0	90.0	88.0	262.0	MARY	A	A	1
4	Jane	75.0	84.0	75.0	229.0	JANE	B	C	3
5	King	65.0	77.0	93.0	230.0	KING	B	C	3
6	Hillary	95.0	55.0	92.0	237.0	HILLARY	B	C	3
7	Ethan	81.0	70.0	98.0	244.0	ETHAN	B	B	2
8	Page	80.0	66.0	90.0	231.0	PAGE	B	C	3

Using applymap

Here we just count the number of characters in each value (length) across the data frame.

In [170]:
```
apply_data_df.applymap(lambda x: len(str(x)))
```

Out[170]:

	Students	Math	Physics	Computer	Total	Students__Upper_Case_Names	Major_Grades	Detailed_Grades	Positions
0	3	4	4	4	5	3	1	1	1
1	5	4	4	4	5	5	1	1	1
2	5	4	4	4	5	5	1	1	1
3	4	4	4	4	5	4	1	1	1
4	4	4	4	4	5	4	1	1	1
5	4	4	4	4	5	4	1	1	1
6	7	4	4	4	5	7	1	1	1
7	5	4	4	4	5	5	1	1	1
8	4	4	4	4	5	4	1	1	1

Let's add 5 to Math course

Note that you can not use applymap to Series only e.g. use applymap on Math only, you'll get an error so we add atleast two columns. The best use-case for this is to use apply function that can take either series or dataframe.

In [171]:
```
apply_data_df[['Math','Computer']].applymap(lambda x: x+5)
```

Out[171]:

	Math	Computer
0	89.0	89.0
1	77.0	83.0
2	90.0	62.0
3	94.0	93.0
4	80.0	80.0
5	70.0	98.0
6	100.0	97.0
7	86.0	103.0
8	85.0	95.0

11. Pandas String Functions

In [172]:
```
titanic_df.head()
```

Out[172]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500	107.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833	171.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250	107.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000	153.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500	108.0500

Convert text to Upper Case

In [173]:
```
titanic_df['Name'].str.upper().head()
```

Out[173]:
```
0                              MR. OWEN HARRIS BRAUND
1    MRS. JOHN BRADLEY (FLORENCE BRIGGS THAYER) CUM...
2                               MISS. LAINA HEIKKINEN
3          MRS. JACQUES HEATH (LILY MAY PEEL) FUTRELLE
4                             MR. WILLIAM HENRY ALLEN
Name: Name, dtype: object
```

Convert text to Lower Case

In [174]:
```
titanic_df['Name'].str.lower().head()
```

Out[174]:
```
0                               mr. owen harris braund
1    mrs. john bradley (florence briggs thayer) cum...
2                                miss. laina heikkinen
3          mrs. jacques heath (lily may peel) futrelle
4                              mr. william henry allen
Name: Name, dtype: object
```

Calculate length of a string

In [175]:
```
titanic_df['Name'].str.len().head()
```

Out[175]:
```
0    22
1    50
2    21
3    43
4    23
Name: Name, dtype: int64
```

Strip white spaces

In [176]:
```
titanic_df['Name'].str.strip().head()
```

Out[176]:
```
0                               Mr. Owen Harris Braund
1    Mrs. John Bradley (Florence Briggs Thayer) Cum...
2                                Miss. Laina Heikkinen
3          Mrs. Jacques Heath (Lily May Peel) Futrelle
4                              Mr. William Henry Allen
Name: Name, dtype: object
```

Split string

In [177]:
```
titanic_df['Name'].str.split(' ').head()
```

Out[177]:
```
0                          [Mr., Owen, Harris, Braund]
1    [Mrs., John, Bradley, (Florence, Briggs, Thaye...
2                            [Miss., Laina, Heikkinen]
3    [Mrs., Jacques, Heath, (Lily, May, Peel), Futr...
4                         [Mr., William, Henry, Allen]
Name: Name, dtype: object
```

Convert strings to categorical numbers

In [178]:
```
titanic_df['Sex'].str.get_dummies().head()
```

Out[178]:

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

Repeat text

In [179]:
```
titanic_df['Sex'].str.repeat(5).head()
```

Out[179]:
```
0              malemalemalemalemale
1    femalefemalefemalefemalefemale
2    femalefemalefemalefemalefemale
3    femalefemalefemalefemalefemale
4              malemalemalemalemale
Name: Sex, dtype: object
```

Count occurrence of certain words

```
In [180]: titanic_df['Name'].str.count('Rev.').sum()
```

```
Out[180]: 6
```

String search startswith and endswith

```
In [181]: titanic_df['Name'].str.startswith('Mrs.').sum()
          titanic_df['Name'].str.endswith('y').sum()
```

```
Out[181]: 51
```

Swap Cases

```
In [182]: titanic_df['Name'].str.swapcase().head()
```

```
Out[182]: 0                               mR. oWEN hARRIS bRAUND
          1    mRS. jOHN bRADLEY (fLORENCE bRIGGS tHAYER) cUM...
          2                                mISS. lAINA hEIKKINEN
          3          mRS. jACQUES hEATH (lILY mAY pEEL) fUTRELLE
          4                               mR. wILLIAM hENRY aLLEN
          Name: Name, dtype: object
```

Filter data with substring

```
In [183]: titanic_df['Name'].str.contains('Mr.').head()
```

```
Out[183]: 0     True
          1     True
          2    False
          3     True
          4     True
          Name: Name, dtype: bool
```

Replace string

```
In [184]: titanic_df['Sex'].str.replace('female','F').head()
```

```
Out[184]: 0    male
          1       F
          2       F
          3       F
          4    male
          Name: Sex, dtype: object
```

Check if text is lower or upper

```
In [185]: titanic_df['Name'].str.islower()
          titanic_df['Name'].str.isupper().head()
```

```
Out[185]: 0    False
          1    False
          2    False
          3    False
          4    False
          Name: Name, dtype: bool
```

Check if value is numeric

In [186]:
```
titanic_df['Name'].str.isnumeric().head()
```

Out[186]:
```
0    False
1    False
2    False
3    False
4    False
Name: Name, dtype: bool
```

12. Mathematical Operations

In [187]:
```
students_score_df = pd.DataFrame(
    {
        "Students": ["Tom", "Peter", "Mary", "Smith"],
        "Reg_No": [1790, 1731, 1780, 1755],
        "Reg_Date": ["15/01/2021", "16/01/2021", "19/01/2021", "27/01/2021"],
        "Math": ["79.00", "67.00", "84.00", "70.00"],
        "Physics": ["60", "70", "50", "90"],
        "Computer": ["65.80", "80", "70", "75"],
    }
)

students_score_df
```

Out[187]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer
0	Tom	1790	15/01/2021	79.00	60	65.80
1	Peter	1731	16/01/2021	67.00	70	80
2	Mary	1780	19/01/2021	84.00	50	70
3	Smith	1755	27/01/2021	70.00	90	75

In [188]:
```
# Check if data types have proper data structure representation if not convert them to proper data types
students_score_df.dtypes
```

Out[188]:
```
Students    object
Reg_No       int64
Reg_Date    object
Math        object
Physics     object
Computer    object
dtype: object
```

In [189]:
```
# Convert data types to proper representation
students_score_df[['Math','Physics','Computer']]=students_score_df[['Math','Physics','Computer']].astype(np.float) # M
ath, Physics and Computer need to be float
students_score_df['Reg_No']=students_score_df['Reg_No'].astype(str) # Reg_No need to object
students_score_df['Reg_Date']=pd.to_datetime(students_score_df['Reg_Date']) # Reg_Date need to be a valid pandas date
```

In [190]:
```
students_score_df.dtypes
```

Out[190]:
```
Students            object
Reg_No              object
Reg_Date    datetime64[ns]
Math               float64
Physics            float64
Computer           float64
dtype: object
```

Scalar Addition

Add a scalar value to every numeric element in the dataframe

In [191]:
```
students_score_df[['Math','Physics','Computer']]=students_score_df[['Math','Physics','Computer']].add(5)
students_score_df.head()
```

Out[191]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer
0	Tom	1790	2021-01-15	84.0	65.0	70.8
1	Peter	1731	2021-01-16	72.0	75.0	85.0
2	Mary	1780	2021-01-19	89.0	55.0	75.0
3	Smith	1755	2021-01-27	75.0	95.0	80.0

Elementwise addition

Add two dataframes elementwise

In [192]:
```
score_1_df=students_score_df[['Math','Physics','Computer']]-90
score_2_df=students_score_df[['Math','Physics','Computer']]-85
```

In [193]:
```
score_1_df
```

Out[193]:

	Math	Physics	Computer
0	-6.0	-25.0	-19.2
1	-18.0	-15.0	-5.0
2	-1.0	-35.0	-15.0
3	-15.0	5.0	-10.0

In [194]:
```
score_2_df
```

Out[194]:

	Math	Physics	Computer
0	-1.0	-20.0	-14.2
1	-13.0	-10.0	0.0
2	4.0	-30.0	-10.0
3	-10.0	10.0	-5.0

In [195]:
```
# add the two dataframes
score_3_df=score_1_df.add(score_2_df)
score_3_df
```

Out[195]:

	Math	Physics	Computer
0	-7.0	-45.0	-33.4
1	-31.0	-25.0	-5.0
2	3.0	-65.0	-25.0
3	-25.0	15.0	-15.0

Subtraction with Scalar value

In [196]:
```
students_score_df
```

Out[196]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer
0	Tom	1790	2021-01-15	84.0	65.0	70.8
1	Peter	1731	2021-01-16	72.0	75.0	85.0
2	Mary	1780	2021-01-19	89.0	55.0	75.0
3	Smith	1755	2021-01-27	75.0	95.0	80.0

In [197]:
```
students_score_df[['Math','Physics','Computer']]=students_score_df[['Math','Physics','Computer']]-50
```

In [198]:
```
students_score_df
```

Out[198]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer
0	Tom	1790	2021-01-15	34.0	15.0	20.8
1	Peter	1731	2021-01-16	22.0	25.0	35.0
2	Mary	1780	2021-01-19	39.0	5.0	25.0
3	Smith	1755	2021-01-27	25.0	45.0	30.0

Elementwise Subtraction

Subtract two dataframes elementwise

In [199]:
```
# score_2_df-score_1_df # Option 1
score_4_df=score_2_df.sub(score_1_df) # Option 2
score_4_df
```

Out[199]:

	Math	Physics	Computer
0	5.0	5.0	5.0
1	5.0	5.0	5.0
2	5.0	5.0	5.0
3	5.0	5.0	5.0

Multiplication with Scalar value

In [200]:
```
students_score_df[['Math','Physics','Computer']]=students_score_df[['Math','Physics','Computer']]*-3
students_score_df
```

Out[200]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer
0	Tom	1790	2021-01-15	-102.0	-45.0	-62.4
1	Peter	1731	2021-01-16	-66.0	-75.0	-105.0
2	Mary	1780	2021-01-19	-117.0	-15.0	-75.0
3	Smith	1755	2021-01-27	-75.0	-135.0	-90.0

Elementwise Multiplication

Multiply two dataframes elementwise

In [201]:
```
# score_1_df * score_2_df # Option 1
score_1_df.mul(score_2_df) # Option 2
```

Out[201]:

	Math	Physics	Computer
0	6.0	500.0	272.64
1	234.0	150.0	-0.00
2	-4.0	1050.0	150.00
3	150.0	50.0	50.00

Division with Scalar value

In [202]:
```
students_score_df[['Math','Physics','Computer']]=students_score_df[['Math','Physics','Computer']]/3
students_score_df
```

Out[202]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer
0	Tom	1790	2021-01-15	-34.0	-15.0	-20.8
1	Peter	1731	2021-01-16	-22.0	-25.0	-35.0
2	Mary	1780	2021-01-19	-39.0	-5.0	-25.0
3	Smith	1755	2021-01-27	-25.0	-45.0	-30.0

Elementwise Division

Divide two dataframes elementwis

In [203]:
```
# score_1_df / score_2_df # Option 1
score_1_df.div(score_2_df) # Option 2
```

Out[203]:

	Math	Physics	Computer
0	6.000000	1.250000	1.352113
1	1.384615	1.500000	-inf
2	-0.250000	1.166667	1.500000
3	1.500000	0.500000	2.000000

Pandas power function

Using ** to raise element to specified power

In [204]:
```
score_1_df**2 # Raise each element to power 2
```

Out[204]:

	Math	Physics	Computer
0	36.0	625.0	368.64
1	324.0	225.0	25.00
2	1.0	1225.0	225.00
3	225.0	25.0	100.00

Using power pow() function

The pow() function calculates the exponential power of dataframe and other, element-wise (binary operator pow). It resembles the ** operator but allows handling of missing values.

In [205]:
```
score_1_df=score_1_df.pow(2)
score_1_df
```

Out[205]:

	Math	Physics	Computer
0	36.0	625.0	368.64
1	324.0	225.0	25.00
2	1.0	1225.0	225.00
3	225.0	25.0	100.00

Elementwise power along specified axis

We can specify axis when performing exponential power of two dataframes or a dataframe and a series

Logarithm on base 2

We can use numpy log function to perform logarithmic operation on dataframe

In [206]:
```
score_1_df['Log2_Computer']=np.log2(score_1_df['Computer'])
score_1_df
```

Out[206]:

	Math	Physics	Computer	Log2_Computer
0	36.0	625.0	368.64	8.526069
1	324.0	225.0	25.00	4.643856
2	1.0	1225.0	225.00	7.813781
3	225.0	25.0	100.00	6.643856

Logarithm on base 10

In [207]:
```
score_1_df['Log10_Computer']=np.log10(score_1_df['Computer'])
score_1_df
```

Out[207]:

	Math	Physics	Computer	Log2_Computer	Log10_Computer
0	36.0	625.0	368.64	8.526069	2.566602
1	324.0	225.0	25.00	4.643856	1.397940
2	1.0	1225.0	225.00	7.813781	2.352183
3	225.0	25.0	100.00	6.643856	2.000000

Natural logarithmic

In [208]:
```
score_1_df['Natural_Log_Computer']=np.log(score_1_df['Computer'])
score_1_df
```

Out[208]:

	Math	Physics	Computer	Log2_Computer	Log10_Computer	Natural_Log_Computer
0	36.0	625.0	368.64	8.526069	2.566602	5.909821
1	324.0	225.0	25.00	4.643856	1.397940	3.218876
2	1.0	1225.0	225.00	7.813781	2.352183	5.416100
3	225.0	25.0	100.00	6.643856	2.000000	4.605170

Pandas aggregate agg function

In [209]:
```
score_1_df
```

Out[209]:

	Math	Physics	Computer	Log2_Computer	Log10_Computer	Natural_Log_Computer
0	36.0	625.0	368.64	8.526069	2.566602	5.909821
1	324.0	225.0	25.00	4.643856	1.397940	3.218876
2	1.0	1225.0	225.00	7.813781	2.352183	5.416100
3	225.0	25.0	100.00	6.643856	2.000000	4.605170

In [210]:
```
score_1_df.agg('mean')
```

Out[210]:
```
Math                   146.500000
Physics                525.000000
Computer               179.660000
Log2_Computer            6.906891
Log10_Computer           2.079181
Natural_Log_Computer     4.787492
dtype: float64
```

Aggregate values a long a specified axis

In [211]:
```
score_1_df.agg('mean',axis=1) # axis=1 implies columnwise
```

Out[211]:
```
0    174.440415
1     97.210112
2    244.430344
3     60.541504
dtype: float64
```

Nesting multiple aggregations

In [212]:
```
score_1_df.agg(['min','max','sum','mean','std','var'])
```

Out[212]:

	Math	Physics	Computer	Log2_Computer	Log10_Computer	Natural_Log_Computer
min	1.00000	25.000000	25.000000	4.643856	1.397940	3.218876
max	324.00000	1225.000000	368.640000	8.526069	2.566602	5.909821
sum	586.00000	2100.000000	718.640000	27.627562	8.316725	19.149967
mean	146.50000	525.000000	179.660000	6.906891	2.079181	4.787492
std	153.89282	529.150262	150.592814	1.696536	0.510708	1.175949
var	23683.00000	280000.000000	22678.195733	2.878233	0.260823	1.382856

Using different agg() functions on each column

In [213]:
```
score_1_df.agg({'Math':['sum','min','max'],'Log2_Computer':['mean'],'Log10_Computer':['std'],'Natural_Log_Computer':[
'var']})
```

Out[213]:

	Math	Log2_Computer	Log10_Computer	Natural_Log_Computer
max	324.0	NaN	NaN	NaN
mean	NaN	6.906891	NaN	NaN
min	1.0	NaN	NaN	NaN
std	NaN	NaN	0.510708	NaN
sum	586.0	NaN	NaN	NaN
var	NaN	NaN	NaN	1.382856

Subtract one year from the date

In [214]:
```
students_score_df['Reg_Date_Less_1_Yr']=students_score_df['Reg_Date']-pd.DateOffset(years=1)
students_score_df
```

Out[214]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer	Reg_Date_Less_1_Yr
0	Tom	1790	2021-01-15	-34.0	-15.0	-20.8	2020-01-15
1	Peter	1731	2021-01-16	-22.0	-25.0	-35.0	2020-01-16
2	Mary	1780	2021-01-19	-39.0	-5.0	-25.0	2020-01-19
3	Smith	1755	2021-01-27	-25.0	-45.0	-30.0	2020-01-27

Subtract one month from the date

In [215]:
```
students_score_df['Reg_Date_Less_1_Mn']=students_score_df['Reg_Date']-pd.DateOffset(months=1)
students_score_df
```

Out[215]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer	Reg_Date_Less_1_Yr	Reg_Date_Less_1_Mn
0	Tom	1790	2021-01-15	-34.0	-15.0	-20.8	2020-01-15	2020-12-15
1	Peter	1731	2021-01-16	-22.0	-25.0	-35.0	2020-01-16	2020-12-16
2	Mary	1780	2021-01-19	-39.0	-5.0	-25.0	2020-01-19	2020-12-19
3	Smith	1755	2021-01-27	-25.0	-45.0	-30.0	2020-01-27	2020-12-27

Subtract one day from the date

In [216]:
```
students_score_df['Reg_Date_Less_1_Day']=students_score_df['Reg_Date']-pd.DateOffset(days=1)
students_score_df
```

Out[216]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer	Reg_Date_Less_1_Yr	Reg_Date_Less_1_Mn	Reg_Date_Less_1_Day
0	Tom	1790	2021-01-15	-34.0	-15.0	-20.8	2020-01-15	2020-12-15	2021-01-14
1	Peter	1731	2021-01-16	-22.0	-25.0	-35.0	2020-01-16	2020-12-16	2021-01-15
2	Mary	1780	2021-01-19	-39.0	-5.0	-25.0	2020-01-19	2020-12-19	2021-01-18
3	Smith	1755	2021-01-27	-25.0	-45.0	-30.0	2020-01-27	2020-12-27	2021-01-26

13. Statistical functions

In [217]:
```
titanic_df.head()
```

Out[217]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500	107.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833	171.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250	107.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000	153.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500	108.0500

Summary statistics

In [218]:
```
titanic_df.describe(include='all') # To also include categorical summary
titanic_df.describe()
```

Out[218]:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
count	887.000000	887.000000	887.000000	887.000000	887.000000	887.00000	887.00000
mean	0.385569	2.305524	29.471443	0.525366	0.383315	32.30542	132.30542
std	0.487004	0.836662	14.121908	1.104669	0.807466	49.78204	49.78204
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.00000	100.00000
25%	0.000000	2.000000	20.250000	0.000000	0.000000	7.92500	107.92500
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.45420	114.45420
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.13750	131.13750
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.32920	612.32920

Show min value

In [219]:
```
titanic_df['Age'].min() # min of age column only
titanic_df.min() # min of every column
```

Out[219]:
```
Survived                                    0
Pclass                                      1
Name                       Capt. Edward Gifford Crosby
Sex                                      female
Age                                        0.42
Siblings/Spouses Aboard                       0
Parents/Children Aboard                       0
Fare                                          0
New_Fare                                    100
dtype: object
```

Show max value

```
In [220]: titanic_df['Age'].max() # max of age column only
          titanic_df.max() # max of every column
```

```
Out[220]: Survived                                                        1
          Pclass                                                          3
          Name                       the Countess. of (Lucy Noel Martha Dyer-Edward...
          Sex                                                          male
          Age                                                            80
          Siblings/Spouses Aboard                                         8
          Parents/Children Aboard                                         6
          Fare                                                      512.329
          New_Fare                                                  612.329
          dtype: object
```

Show mode of values

```
In [221]: titanic_df['Age'].mode() # mode of Age
```

```
Out[221]: 0    22.0
          dtype: float64
```

Show median of values

```
In [222]: titanic_df['Fare'].median() # Median of Fare
```

```
Out[222]: 14.4542
```

Show sum of values

In [223]:
```
titanic_df['Fare'].sum() # max of Fare
```

Out[223]:
```
28654.907699999996
```

Show frequency of each category

In [224]:
```
titanic_df['Pclass'].value_counts() # freequency of passengers in each class
```

Out[224]:
```
3    487
1    216
2    184
Name: Pclass, dtype: int64
```

Calculate mean

In [225]:
```
titanic_df['Age'].mean()
```

Out[225]:
```
29.471443066516347
```

Calculate standard deviation

In [226]:
```
titanic_df['Age'].std() # std for age only
titanic_df.std() # std for all columns
```

Out[226]:
```
Survived                  0.487004
Pclass                    0.836662
Age                      14.121908
Siblings/Spouses Aboard   1.104669
Parents/Children Aboard   0.807466
Fare                     49.782040
New_Fare                 49.782040
dtype: float64
```

Show Variance

In [227]:
```
titanic_df['Age'].var() # variance for age column only
titanic_df.var() # variance for all numeric columns
```

Out[227]:
```
Survived                     0.237173
Pclass                       0.700003
Age                        199.428297
Siblings/Spouses Aboard      1.220293
Parents/Children Aboard      0.652001
Fare                      2478.251546
New_Fare                  2478.251546
dtype: float64
```

Show Covariance

In [228]:
```
titanic_df[['Age','Fare']].cov() # Covariance of Age and Fare
titanic_df.cov() # Covariance for entire dataframe
```

Out[228]:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
Survived	0.237173	-0.137121	-0.410343	-0.019950	0.031497	6.210808	6.210808
Pclass	-0.137121	0.700003	-4.625577	0.078584	0.013681	-22.862898	-22.862898
Age	-0.410343	-4.625577	199.428297	-4.643648	-2.209222	78.968988	78.968988
Siblings/Spouses Aboard	-0.019950	0.078584	-4.643648	1.220293	0.369498	8.734998	8.734998
Parents/Children Aboard	0.031497	0.013681	-2.209222	0.369498	0.652001	8.661314	8.661314
Fare	6.210808	-22.862898	78.968988	8.734998	8.661314	2478.251546	2478.251546
New_Fare	6.210808	-22.862898	78.968988	8.734998	8.661314	2478.251546	2478.251546

Show Correlation

Correlation Measures the relationship between two variables

1. Pearson Correlation

Measures the linear relationship between two variables. Pearson correlation coefficient is the default correlation method in Pandas Data Frame. NOTE: Pearson Correlation assumes that the data is normally distributed. It's sensitive to outliers

In [229]:
```
titanic_df.corr(method='pearson')
titanic_df.corr() # Or don't specifiy since it's the default
```

Out[229]:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
Survived	1.000000	-0.336528	-0.059665	-0.037082	0.080097	0.256179	0.256179
Pclass	-0.336528	1.000000	-0.391492	0.085026	0.020252	-0.548919	-0.548919
Age	-0.059665	-0.391492	1.000000	-0.297669	-0.193741	0.112329	0.112329
Siblings/Spouses Aboard	-0.037082	0.085026	-0.297669	1.000000	0.414244	0.158839	0.158839
Parents/Children Aboard	0.080097	0.020252	-0.193741	0.414244	1.000000	0.215470	0.215470
Fare	0.256179	-0.548919	0.112329	0.158839	0.215470	1.000000	1.000000
New_Fare	0.256179	-0.548919	0.112329	0.158839	0.215470	1.000000	1.000000

2. Spearman Rank Correlation

Measures the monotonic relationship between two variables. Does not assume normal distribution of the dataset. Has a growth rate of O(nlogn)

In [230]:
```
titanic_df.corr(method='spearman')
```

Out[230]:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
Survived	1.000000	-0.337648	-0.030265	0.086571	0.136530	0.322264	0.322264
Pclass	-0.337648	1.000000	-0.387982	-0.040348	-0.020617	-0.688234	-0.688234
Age	-0.030265	-0.387982	1.000000	-0.199269	-0.254234	0.156062	0.156062
Siblings/Spouses Aboard	0.086571	-0.040348	-0.199269	1.000000	0.449198	0.445980	0.445980
Parents/Children Aboard	0.136530	-0.020617	-0.254234	0.449198	1.000000	0.409202	0.409202
Fare	0.322264	-0.688234	0.156062	0.445980	0.409202	1.000000	1.000000
New_Fare	0.322264	-0.688234	0.156062	0.445980	0.409202	1.000000	1.000000

3. Kendall Rank Correlation

It measures the monotonic relationship between two variables. It does not assume normal distribution of the data. It has a growth rate of O(n^2) hence tends to be abit slower on large dataset.

In [231]:
```
titanic_df.corr(method='kendall')
```

Out[231]:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
Survived	1.000000	-0.321558	-0.024993	0.083671	0.132233	0.264998	0.264998
Pclass	-0.321558	1.000000	-0.308253	-0.037085	-0.018995	-0.573648	-0.573648
Age	-0.024993	-0.308253	1.000000	-0.156090	-0.200868	0.107466	0.107466
Siblings/Spouses Aboard	0.083671	-0.037085	-0.156090	1.000000	0.424367	0.357309	0.357309
Parents/Children Aboard	0.132233	-0.018995	-0.200868	0.424367	1.000000	0.329609	0.329609
Fare	0.264998	-0.573648	0.107466	0.357309	0.329609	1.000000	1.000000
New_Fare	0.264998	-0.573648	0.107466	0.357309	0.329609	1.000000	1.000000

Calculate Kurtosis

In [232]:
```
titanic_df.kurtosis()
```

Out[232]:
```
Survived                 -1.782183
Pclass                   -1.288638
Age                       0.292559
Siblings/Spouses Aboard  17.797537
Parents/Children Aboard   9.723066
Fare                     33.264605
New_Fare                 33.264605
dtype: float64
```

Calculate Skew

```
In [233]: titanic_df.skew()
```

```
Out[233]: Survived                  0.470999
          Pclass                   -0.623409
          Age                       0.447189
          Siblings/Spouses Aboard   3.686760
          Parents/Children Aboard   2.741198
          Fare                      4.777671
          New_Fare                  4.777671
          dtype: float64
```

Compute Percent change

Calculates the percent change over a given number of periods. Handle missing values (Nulls) before computing the percent change).

```
In [234]: titanic_df['Fare'].pct_change(periods=3)
```

```
Out[234]: 0           NaN
          1           NaN
          2           NaN
          3      6.324138
          4     -0.887070
                   ...
          882    0.238095
          883    3.255319
          884   -0.194850
          885    1.307692
          886   -0.741667
          Name: Fare, Length: 887, dtype: float64
```

Rank

ranks the data and shows the ties in data values

In [235]:
```
titanic_df.rank().head()
```

Out[235]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	New_Fare
0	273.0	644.0	602.0	601.0	277.0	709.0	337.5	77.0	77.0
1	716.5	108.5	823.0	157.5	667.5	709.0	337.5	785.0	785.0
2	716.5	644.0	172.0	157.5	392.0	302.5	337.5	229.5	229.5
3	716.5	108.5	814.0	157.5	615.0	709.0	337.5	744.0	744.0
4	273.0	644.0	733.0	601.0	615.0	302.5	337.5	261.0	261.0

14. Window Functions

Lag in Pandas

It's best practice to sort values first based on a key column such as date

In [236]:
```
windows_data_df = pd.DataFrame(
    {
        "Students": ["Tom", "Peter", "Mary", "Smith"],
        "Reg_No": ["1790","1731","1780", "1755"],
        "Reg_Date": ["15/01/2021", "13/01/2021", "14/01/2021", "27/01/2021"],
        "Math": [79.00, 67.00, 84.00, 70.00],
        "Physics": [60, 70, 50, 90],
        "Computer": [65.80, 80, 70, 75],
    }
)

windows_data_df
```

Out[236]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer
0	Tom	1790	15/01/2021	79.0	60	65.8
1	Peter	1731	13/01/2021	67.0	70	80.0
2	Mary	1780	14/01/2021	84.0	50	70.0
3	Smith	1755	27/01/2021	70.0	90	75.0

In [237]:
```
windows_data_df=windows_data_df.sort_values(by='Reg_Date', ascending=False)
windows_data_df['Previous_Computer'] = windows_data_df['Computer'].shift(1)
windows_data_df
```

Out[237]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer	Previous_Computer
3	Smith	1755	27/01/2021	70.0	90	75.0	NaN
0	Tom	1790	15/01/2021	79.0	60	65.8	75.0
2	Mary	1780	14/01/2021	84.0	50	70.0	65.8
1	Peter	1731	13/01/2021	67.0	70	80.0	70.0

Lead in Pandas

It's best practice to sort values first based on a key column such as date

In [238]:
```
windows_data_df=windows_data_df.sort_values(by='Reg_Date', ascending=False)
windows_data_df['Next_Computer'] = windows_data_df['Computer'].shift(-1)
windows_data_df
```

Out[238]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer	Previous_Computer	Next_Computer
3	Smith	1755	27/01/2021	70.0	90	75.0	NaN	65.8
0	Tom	1790	15/01/2021	79.0	60	65.8	75.0	70.0
2	Mary	1780	14/01/2021	84.0	50	70.0	65.8	80.0
1	Peter	1731	13/01/2021	67.0	70	80.0	70.0	NaN

Rolling window

Generic fixed or variable sliding window over the values.

In [239]:
```
'''Look at how we are calculating the Rolling_Window_Physics column. We're going back and adding previous
    number to the current. In this case our window=2'''
windows_data_df['Rolling_Window_Physics']=windows_data_df['Physics'].rolling(window=2).sum()
windows_data_df
```

Out[239]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer	Previous_Computer	Next_Computer	Rolling_Window_Physics
3	Smith	1755	27/01/2021	70.0	90	75.0	NaN	65.8	NaN
0	Tom	1790	15/01/2021	79.0	60	65.8	75.0	70.0	150.0
2	Mary	1780	14/01/2021	84.0	50	70.0	65.8	80.0	110.0
1	Peter	1731	13/01/2021	67.0	70	80.0	70.0	NaN	120.0

Specifying Weights in the Windows

Passing win_type to .rolling generates a generic rolling window computation, that is weighted according the win_type. Refer to documentation for more details https://pandas-docs.github.io/pandas-docs-travis/user_guide/computation.html#window-functions (https://pandas-docs.github.io/pandas-docs-travis/user_guide/computation.html#window-functions)

In [240]:
```
windows_data_df['Rolling_Window_Physics']=windows_data_df['Physics'].rolling(window=2, win_type='triang').sum()
windows_data_df
```

Out[240]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer	Previous_Computer	Next_Computer	Rolling_Window_Physics
3	Smith	1755	27/01/2021	70.0	90	75.0	NaN	65.8	NaN
0	Tom	1790	15/01/2021	79.0	60	65.8	75.0	70.0	75.0
2	Mary	1780	14/01/2021	84.0	50	70.0	65.8	80.0	55.0
1	Peter	1731	13/01/2021	67.0	70	80.0	70.0	NaN	60.0

Expanding window

Accumulating window over the values. An expanding window yields the value of an aggregation statistic with all the data available up to that point in time.

In [241]:
```
'''Expanding window with min_periods=1 with sum function resembes cumulative sum.
    Trying using other aggregation functions such as mean etc.'''
windows_data_df['Expanding_Window_Physics']=windows_data_df['Physics'].expanding(min_periods=1).mean()
windows_data_df
```

Out[241]:

	Students	Reg_No	Reg_Date	Math	Physics	Computer	Previous_Computer	Next_Computer	Rolling_Window_Physics	Expandin
3	Smith	1755	27/01/2021	70.0	90	75.0	NaN	65.8	NaN	90.000000
0	Tom	1790	15/01/2021	79.0	60	65.8	75.0	70.0	75.0	75.000000
2	Mary	1780	14/01/2021	84.0	50	70.0	65.8	80.0	55.0	66.666667
1	Peter	1731	13/01/2021	67.0	70	80.0	70.0	NaN	60.0	67.500000

Exponentially Weighted window

Exponentially Weighted window: Accumulating and exponentially weighted window over the values. An exponentially weighted window is similar to an expanding window but with each prior point being exponentially weighted down relative to the current point. Available EW functions are mean(), var(), std(), corr(), cov()

In [242]:
```
# Let's load our sales data
clv_df=pd.read_csv('clv.csv')
```

In [243]:
```
clv_df.head()
```

Out[243]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01/12/2010	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	01/12/2010	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01/12/2010	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01/12/2010	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01/12/2010	3.39	17850.0	United Kingdom

In [244]:
```
# Let's created a sales column by multiplying Quantity by UnitPrice
clv_df['sales']=clv_df['Quantity']*clv_df['UnitPrice']
clv_df.head()
```

Out[244]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	sales
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01/12/2010	2.55	17850.0	United Kingdom	15.30
1	536365	71053	WHITE METAL LANTERN	6	01/12/2010	3.39	17850.0	United Kingdom	20.34
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01/12/2010	2.75	17850.0	United Kingdom	22.00
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01/12/2010	3.39	17850.0	United Kingdom	20.34
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01/12/2010	3.39	17850.0	United Kingdom	20.34

In [245]:
```
# Let's order our data in ascending order based on InvoiceDate
clv_df=clv_df.sort_values(by='InvoiceDate',ascending=True)
clv_df.head()
```

Out[245]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	sales
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01/12/2010	2.55	17850.0	United Kingdom	15.30
2066	536557	84029E	RED WOOLLY HOTTIE WHITE HEART.	1	01/12/2010	3.75	17841.0	United Kingdom	3.75
2067	536557	22678	FRENCH BLUE METAL DOOR SIGN 3	3	01/12/2010	1.25	17841.0	United Kingdom	3.75
2068	536557	22686	FRENCH BLUE METAL DOOR SIGN No	1	01/12/2010	1.25	17841.0	United Kingdom	1.25
2069	536557	22468	BABUSHKA LIGHTS STRING OF 10	1	01/12/2010	6.75	17841.0	United Kingdom	6.75

exponentially weighted moving average

Exponentially weighted moving average gives more weight to recent observations, which makes it powerful to capture recent trends more quickly.

In [246]:
```
### Exponential Moving Average on sales
clv_df['5_day_Sales_EWM'] = clv_df['sales'].ewm(span=5).mean()
clv_df.head(10)
```

Out[246]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	sales	5_day_Sales_EWM
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01/12/2010	2.55	17850.0	United Kingdom	15.30	15.300000
2066	536557	84029E	RED WOOLLY HOTTIE WHITE HEART.	1	01/12/2010	3.75	17841.0	United Kingdom	3.75	8.370000
2067	536557	22678	FRENCH BLUE METAL DOOR SIGN 3	3	01/12/2010	1.25	17841.0	United Kingdom	3.75	6.181579
2068	536557	22686	FRENCH BLUE METAL DOOR SIGN No	1	01/12/2010	1.25	17841.0	United Kingdom	1.25	4.133077
2069	536557	22468	BABUSHKA LIGHTS STRING OF 10	1	01/12/2010	6.75	17841.0	United Kingdom	6.75	5.137678
2070	536557	85232B	SET OF 3 BABUSHKA STACKING TINS	1	01/12/2010	4.95	17841.0	United Kingdom	4.95	5.069098
2071	536557	21479	WHITE SKULL HOT WATER BOTTLE	5	01/12/2010	3.75	17841.0	United Kingdom	18.75	9.912895
2072	536557	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	2	01/12/2010	3.75	17841.0	United Kingdom	7.50	9.075940
2073	536557	22837	HOT WATER BOTTLE BABUSHKA	5	01/12/2010	4.65	17841.0	United Kingdom	23.25	13.926809
2074	536557	22112	CHOCOLATE HOT WATER BOTTLE	2	01/12/2010	4.95	17841.0	United Kingdom	9.90	12.560851