In [ ]:

```
# Import the needed library
import pandas as pd
```

Create Canned data

Canned data is hard coded within the program

Create a key:value collection of series to use to populate the dataframe for testing

In [ ]:

```
data = {'Month': pd.Series(['January', 'February', 'March', 'April', 'May','June', 'July
        'Rainfall': pd.Series([1.65,1.25,1.94, 2.75, 3.14, 3.65, 5.05, 1.50, 1.33, 0.07, 
        }
```

In [ ]:

```
# creates a Pandas DataFrame with two columns 'Month' and 'Rainfall',
df = pd.DataFrame(data)
df.shape
```

Out[ ]:

```
(12, 2)
```

In [ ]:

```
# print("Our data frame:")
# print(df, "\n")
df
```

Out[ ]:

	Month	Rainfall
0	January	1.65
1	February	1.25
2	March	1.94
3	April	2.75
4	May	3.14
5	June	3.65
6	July	5.05
7	August	1.50
8	September	1.33
9	October	0.07
10	November	0.50
11	December	2.30

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Month     12 non-null     object 
 1   Rainfall  12 non-null     float64
dtypes: float64(1), object(1)
memory usage: 320.0+ bytes
```

Pandas Series is a single dimension array

Pandas dataframe is a two-dimensional array, like a spreadsheet. Our df consists of 2 rows of series(months and rainfall)

In [ ]:

```
# add 1 row to the above data, with NAN and zero values
data_1 = {'Month': pd.Series(['January', 'February', 'March', 'April', 'May','June', 'Ju
       'Rainfall': pd.Series([1.65,1.25,1.94, 2.75, 3.14, 3.65, 5.05, 1.50, 1.33, 0.07, 
       'Temperature': pd.Series([3,10,15,20,75,"",30,1,33," ",32,7])
       }
```

In [ ]:

```
#change to df
df = pd.DataFrame(data_1)

#SAVE df as a csv file on the same path as ""Processing data with Python.ipynb"
df.to_csv('rainfall.csv', index = 0)

#read a .csv file
df_rainfall = pd.read_csv('rainfall.csv')
df_rainfall
```

Out[ ]:

	Month	Rainfall	Temperature
0	January	1.65	3
1	February	1.25	10
2	March	1.94	15
3	April	2.75	20
4	May	3.14	75
5	June	3.65	NaN
6	July	5.05	30
7	August	1.50	1
8	September	1.33	33
9	October	0.07	
10	November	0.50	32
11	December	2.30	7

In [ ]:

```
#to read .json file
df_json = pd.read_json('data.json')
```

```
print("Our data frame from JSON file:")
print(df_json, "\n")
```

```
Our data frame from JSON file:
        Month  Rainfall  Temperature
0     January     1.650          3.0
1    February     1.250         10.0
2       March     1.940         15.0
3       April     2.750         20.0
4         May     2.750         25.0
5        June     3.645         24.0
6        July     5.500         30.0
7      August     1.000          1.0
8   September     1.300         33.0
9     October       NaN          NaN
10   November     0.500         32.0
11   December     2.300          2.3
```

Cleaning Data:

One of the most important tasks in processing data.

Data needs to be consistent to be reliably analyzed.

Cleaning involves parsing the data detecting 'bad' or missing data

In [ ]:

```
# October is NaN value, so
# To not break the algorithm we will replace NAN with the average temperature value
# Calculate the average temperature value
average_temp = df_json['Temperature'].mean()
average_temp
```

Out[ ]:

```
17.754545454545454
```

In [ ]:

```
# Replace NaN value by  rounded average_temp value
df_zeros = df_json.fillna(round(average_temp,0))
print("Our data with zerod values: ")
print(df_zeros)
#Zero can skew the data so we should remove invalid data
# so, we will not use this data later on
```

```
Our data with zerod values: 
        Month  Rainfall  Temperature
0     January     1.650          3.0
1    February     1.250         10.0
2       March     1.940         15.0
3       April     2.750         20.0
4         May     2.750         25.0
5        June     3.645         24.0
6        July     5.500         30.0
7      August     1.000          1.0
8   September     1.300         33.0
9     October    18.000         18.0
10   November     0.500         32.0
11   December     2.300          2.3
```

```
In [ ]: #remove rows with the missing values
        df_cleaned = df_zeros.dropna()
        print("Our data with dropped values: \n", df_cleaned)
```

```
Our data with dropped values: 
         Month  Rainfall  Temperature
0     January     1.650          3.0
1    February     1.250         10.0
2       March     1.940         15.0
3       April     2.750         20.0
4         May     2.750         25.0
5        June     3.645         24.0
6        July     5.500         30.0
7      August     1.000          1.0
8   September     1.300         33.0
9     October    18.000         18.0
10   November     0.500         32.0
11   December     2.300          2.3
```

```
In [ ]: #create a count of all rows containg Nans to check data before cleaning
        count = 0
        for index, row in df_json.iterrows():
            if any(row.isnull()):
                count = count + 1

        print("\n JSON file have",  str(count), "rows with Nans")
```

```
 JSON file have 1 rows with Nans
```

```
In [ ]: #check the number of NAN rows in the cleaned data
        count = 0
        for index, row in df_cleaned.iterrows():
            if any(row.isnull()):
                count = count + 1

        print("\n Number of rows with Nans: " + str(count))
```

```
 Number of rows with Nans: 0
```

```
In [ ]: df_cleaned = df_clean.sort_index()
        df_cleaned
```

Out[ ]:

	Month	Rainfall	Temperature
0	January	1.650	3.0
1	February	1.250	10.0
2	March	1.940	15.0
3	April	2.750	20.0
4	May	2.750	25.0
5	June	3.645	24.0
6	July	5.500	30.0
7	August	1.000	1.0
8	September	1.300	33.0
9	October	18.000	18.0
10	November	0.500	32.0
11	December	2.300	2.3

Statistical Analysis

Mean = the average of a set of numbers.

Median = The middle calue in a sorted set of numbers.

Standard deviation = How much each value differs from the mean. Can be used to detect outliers.

Mode = The most common value in a list of data.

Pandas easily perform these functions!

In [ ]:

```
print("Mean: ")
print(df_cleaned.mean())

print("              ")
print("Median : ")
print(df_cleaned.median())

#print("              ")
print("\n", "Standard Deviation: ")
print(df_cleaned.std())

#The mode here is wrong but will do it for explanation purposes
#rainfall has repeated value of 2.75
#temperature are all unique
print("\n", "Mode : ")
print(df_cleaned.mode())
```

```
Mean:
Rainfall        3.54875
Temperature    17.77500
dtype: float64

Median :
Rainfall        2.12
Temperature    19.00
dtype: float64

 Standard Deviation:
Rainfall        4.746445
Temperature    11.626312
dtype: float64

 Mode :
        Month  Rainfall  Temperature
0       April      2.75          1.0
1      August       NaN          2.3
2    December       NaN          3.0
3    February       NaN         10.0
4     January       NaN         15.0
5        July       NaN         18.0
6        June       NaN         20.0
7       March       NaN         24.0
8         May       NaN         25.0
9    November       NaN         30.0
10    October       NaN         32.0
11  September       NaN         33.0
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_11364\1655336313.py:2: FutureWarning: Droppin
g of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; i
n a future version this will raise TypeError.  Select only valid columns before calling t
he reduction.
  print(df_cleaned.mean())
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_11364\1655336313.py:6: FutureWarning: Droppin
g of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; i
n a future version this will raise TypeError.  Select only valid columns before calling t
he reduction.
  print(df_cleaned.median())
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_11364\1655336313.py:10: FutureWarning: Droppi
ng of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated;
in a future version this will raise TypeError.  Select only valid columns before calling
the reduction.
  print(df_cleaned.std())
```

```
In [ ]: df_cleaned.describe()
```

Out[ ]:

	Rainfall	Temperature
count	12.000000	12.000000
mean	3.548750	17.775000
std	4.746445	11.626312
min	0.500000	1.000000
25%	1.287500	8.250000
50%	2.120000	19.000000
75%	2.973750	26.250000
max	18.000000	33.000000

Selecting Parts of a Dataframe

Indexing

Select single columns using a column name(temperature). Returns a series.

Example: df_clean['Temperature']

Select multiple columns using column names. Must specify a list of column names.

Example: df_clean[['Temperature', 'Rainfall']]

iloc and loc

Select a certain row number using iloc:

Example: print("Third row \n", df_clean.iloc[2])

Select a certain row using a certain value:

Example: print("\n Third row \n", dfIndexed.loc['March']);

In [ ]:

```
print("Temperature column: \n", df_cleaned['Temperature'])
print("\n Temperature and Rainfall column: \n",df_cleaned[['Temperature', 'Rainfall']])

print("\n Third row \n", df_cleaned.iloc[2])
```

```
Temperature column:
 0      3.0
1     10.0
2     15.0
3     20.0
4     25.0
5     24.0
6     30.0
7      1.0
8     33.0
9     18.0
10    32.0
11     2.3
Name: Temperature, dtype: float64

 Temperature and Rainfall column:
     Temperature  Rainfall
0           3.0     1.650
1          10.0     1.250
2          15.0     1.940
3          20.0     2.750
4          25.0     2.750
5          24.0     3.645
6          30.0     5.500
7           1.0     1.000
8          33.0     1.300
9          18.0    18.000
10         32.0     0.500
11          2.3     2.300

 Third row
 Month           March
Rainfall        1.94
Temperature     15.0
Name: 2, dtype: object
```

In [ ]:

```
#To use loc, we require a properly indexed datafram
index = df_cleaned['Month']
dfIndexed = df_cleaned.set_index(index)

print("\n Third row \n", dfIndexed.loc['March'])
```

```
 Third row
 Month           March
Rainfall        1.94
Temperature     15.0
Name: March, dtype: object
```

In [ ]:

```
#Print the rainfall and mean for the first 4 months
rainfall = df_cleaned['Rainfall'][0:4]
print(rainfall, "\n")
print("The mean of rainfall is:", round(rainfall.mean(),2))
```

```
0    1.65
1    1.25
2    1.94
3    2.75
Name: Rainfall, dtype: float64

The mean of rainfall is: 1.9
```

In [ ]:

```
#Print the rainfall and mean for the first few months
print("\n Just Temperature and rainfall data ")
df_TempRain = df_cleaned[['Temperature','Rainfall']]
print(df_TempRain, "\n")
print("The mean of Temperature and Rainfall is: \n", df_TempRain.mean(), "\n")
```

```
 Just Temperature and rainfall data 
    Temperature  Rainfall
0           3.0     1.650
1          10.0     1.250
2          15.0     1.940
3          20.0     2.750
4          25.0     2.750
5          24.0     3.645
6          30.0     5.500
7           1.0     1.000
8          33.0     1.300
9          18.0    18.000
10         32.0     0.500
11          2.3     2.300 

The mean of Temperature and Rainfall is: 
 Temperature    17.77500
Rainfall        3.54875
dtype: float64 
```

Many thanks to MarwaEshra and Instructor (David Dalsveen)