Pandas Guide

Pandas (Basic to Intermediate) full guide with practical examples

ihiteshmodi

Pandas (Basic to Intermediate) full guide with practical examples

INDEX

ihiteshmodi

Pandas Guide

Indexers (Loc And iLoc)

ihiteshmodi

Indexers (Loc and iLoc in Python)

Loc and Iloc are called "indexers"

Lets learn about what they are, how they work and how are they different from each other

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

```
In [5]: df = pd.read_csv("Heart_disease_details.csv")
```

```
In [7]: df.head(5)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-r
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None
3	William Thompson	Male	62	1	1	1	145	90	100	1	...	Shows a 80% blockage in the left circumflex co...	0	1	None

ihiteshmodi

Indexers (Loc and iLoc in Python)

```
# The most important thing to remember for loc and iloc indexers is, Argument 1 is for rows and argument 2 is for columns


# Lets start by seeing how indexers apply to rows
```

```
In [ ]:
```

```
In [ ]:
```

```
Loc and lloc by defauly apply on index only!
when we want to see only the first and second row, use this
```

```
In [10]: df.loc[[0,1]] #loc works with integers as well as strings
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None

```
2 rows x 49 columns
```

```
In [11]: df.iloc[[0,1]] #iloc works with integers only
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None

```
2 rows x 49 columns
```

```
# Lets see how indexers apply to columns
```

```
In [37]: df.loc[[0,1,2,3],["Name", "Gender", "Age"]]
```

	Name	Gender	Age
0	Jane Doe	Female	55
1	Mark Johnson	Male	57
2	Emily Davis	Female	60
3	William Thompson	Male	62

```
## For Iloc, Use column number instead of column name
```

```
In [38]: df.iloc[[0,1,2,3], [0,1,2]]
```

	Gender	Name	Age
0	Female	Jane Doe	55
1	Male	Mark Johnson	57
2	Female	Emily Davis	60
3	Male	William Thompson	62

You can also use slicing with loc and iloc

In [20]:
```
df.loc[0:3]
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-r
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None
3	William Thompson	Male	62	1	1	1	145	90	100	1	...	Shows a 80% blockage in the left circumflex co...	0	1	None

4 rows x 49 columns

In [21]:
```
df.iloc[0:3]
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None

3 rows x 49 columns

Indexers (Loc and iLoc in Python)

```
# Fun Fact! You can use slicing with columns as well
```

```
In [39]: df.loc[0:6, "Name":"Diastolic"]
```

	Name	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic
0	Jane Doe	55	1	1	1	140	90
1	Mark Johnson	57	1	1	1	150	80
2	Emily Davis	60	1	1	1	130	85
3	William Thompson	62	1	1	1	145	90
4	Ashley Johnson	58	1	1	1	135	80
5	Brian Brown	55	1	1	1	150	95
6	Emily Davis	60	1	1	1	145	90

```
## For  Iloc, Use column number instead of column name
```

```
In [40]: df.iloc[0:6, 0:8]
```

	Gender	Name	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic
0	Female	Jane Doe	55	1	1	1	140	90
1	Male	Mark Johnson	57	1	1	1	150	80
2	Female	Emily Davis	60	1	1	1	130	85
3	Male	William Thompson	62	1	1	1	145	90
4	Female	Ashley Johnson	58	1	1	1	135	80
5	Male	Brian Brown	55	1	1	1	150	95

Indexers work on index only! How to apply loc and iloc to specific columns?

If we want to use loc or iloc, We will have to first set the required column as index.

In [27]:
```
df.set_index("Gender", inplace = True)
```

In [29]:
```
df.index
```

```
Index(['Female', 'Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male',
       'Female', 'Male',
       ...
       'Male', 'Female', 'Female', 'Male', 'Female', 'Female', 'Male',
       'Female', 'Male', 'Female'],
      dtype='object', name='Gender', length=334)
```

We see the "Gender" column has been set as the index, (name = "Gender" shows us which column is currently set as the index)

In [31]:
```
df.loc["Female"]
```

Gender	Name	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	Cholesterol level (mg/dL)	...	Cardiac CT	Obesity	Murm
Female	Jane Doe	55	1	1	1	140	90	100	1	220	...	Shows a 50% blockage in the left anterior desc...	0	1
Female	Emily Davis	60	1	1	1	130	85	95	1	230	...	Shows a 75% blockage in the left anterior desc...	0	1
Female	Ashley Johnson	58	1	1	1	135	80	105	1	220	...	Shows a 90% blockage in the right coronary artery	0	1

3 rows x 48 columns

We see the values in the "Gender" Column are "Female" Only!

```
# Bonus for sticking till the end
```

```
## Can we use iLoc even when a string column is set as index?
```

In [32]:
```
df.iloc[0:3]
'''
We can still use iloc to filter by index numbers instead of currently set index
However, if we use loc, currently set index will be used like in example above
'''
```

Gender	Name	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	Cholesterol level (mg/dL)	...	Cardiac CT	Obesity	Murm
Female	Jane Doe	55	1	1	1	140	90	100	1	220	...	Shows a 50% blockage in the left anterior desc...	0	1
Male	Mark Johnson	57	1	1	1	150	80	110	1	210	...	Shows a 60% blockage in the right coronary artery	0	1
Female	Emily Davis	60	1	1	1	130	85	95	1	230	...	Shows a 75% blockage in the left anterior desc...	0	1

3 rows x 48 columns

ihiteshmodi

Pandas Guide

Filtering / Masking

ihiteshmodi

Before we start this lesson, The most imp thing to know is about Booleans "AND" & "OR"

AND in pandas is written as "&"

OR in pandas is written as "|"

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

In [2]: df = pd.read_csv("Heart_disease_details.csv")

In [3]: # This is how our dataset looks like
        df.head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None

3 rows x 49 columns

ihiteshmodi

Filtering / Masking in Pandas - Python

Lets look at only the values where first name is Jane Doe without using a mask

In [4]:
```
df[df["Name"] == "Jane Doe"].head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
8	Jane Doe	Female	55	1	1	1	130	80	110	1	...	Shows a 60% blockage in the left main coronary...	0	1	None
62	Jane Doe	Female	42	1	1	1	120	70	80	1	...	Shows a severe dilatation in the left ventricle	0	1	None

3 rows x 49 columns

ihiteshmodi

Now, look at only the values where first name is Jane Doe using a mask

In [5]:
```
'''
You can name this anything, I usually name it filt or mask
You can also give a descriptive name like "Jane Only Filter"
'''


#Creating the mask
mask = (df["Name"] == "Jane Doe")

#applying the mask
df[mask].head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
8	Jane Doe	Female	55	1	1	1	130	80	110	1	...	Shows a 60% blockage in the left main coronary...	0	1	None
62	Jane Doe	Female	42	1	1	1	120	70	80	1	...	Shows a severe dilatation in the left ventricle	0	1	None

3 rows x 49 columns

Applying the mask using Loc

In [6]:
```
#Creating the mask
mask = (df["Name"] == "Jane Doe")

#applying the mask
df.loc[mask].head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
8	Jane Doe	Female	55	1	1	1	130	80	110	1	...	Shows a 60% blockage in the left main coronary...	0	1	None
62	Jane Doe	Female	42	1	1	1	120	70	80	1	...	Shows a severe dilatation in the left ventricle	0	1	None

3 rows × 49 columns

Using loc to call only required columns with filter applied

In [7]:
```
#Creating the mask
mask = (df["Name"] == "Jane Doe")

#applying the mask
df.loc[mask, "Name" : "Systolic"].head(3)  #If you dont know how to use loc, Refer to my Loc Guide
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic
0	Jane Doe	Female	55	1	1	1	140
8	Jane Doe	Female	55	1	1	1	130
62	Jane Doe	Female	42	1	1	1	120

Filtering / Masking in Pandas - Python

Now, Lets apply multiple filters

In [ ]:

We will filter for all individuals with name "jane Doe" and Heart Rate (bpm) greater than or equal to 100

In [8]:
```
#Creating the mask
mask = (df["Name"] == "Jane Doe") & (df["Heart rate (bpm)"] >= 100)

#applying the mask
df.loc[mask]  #If you dont know how to use loc, Refer to my Loc Guide
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-r
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	Non
8	Jane Doe	Female	55	1	1	1	130	80	110	1	...	Shows a 60% blockage in the left main coronary...	0	1	Non
162	Jane Doe	Female	55	1	0	1	120	80	100	1	...	Shows thickening of the pericardium	0	0	Non
325	Jane Doe	Female	72	1	1	1	130	80	100	1	...	Shows thickening of the pericardium	0	1	Non

4 rows x 49 columns

When using multiple filters wrapping each one in parenthesis in important

ihiteshmodi

In [ ]:

Also, We can only use symbols "&" (Symbol for Boolean "AND") and "|" (Symbol for Boolean "OR") in pandas not the Booleans "AND" and "OR" directly

In [ ]:

Filtering / Masking in Pandas - Python

We will filter for all individuals with either name "jane Doe" or Heart Rate (bpm) greater than or equal to 100

In [9]:
```
#Creating the mask
mask = (df["Name"] == "Jane Doe") | (df["Heart rate (bpm)"] >= 100)

#applying the mask
df.loc[mask].head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-r
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
3	William Thompson	Male	62	1	1	1	145	90	100	1	...	Shows a 80% blockage in the left circumflex co...	0	1	None

3 rows x 49 columns

ihiteshmodi

Now, Lets learn how to get opposite of a filter

Using the filter normally gives us filtered results

In [ ]:

In [10]:
```
#Creating the mask
mask = (df["Name"] == "Jane Doe")

#applying the mask
df.loc[mask].head(2)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
8	Jane Doe	Female	55	1	1	1	130	80	110	1	...	Shows a 60% blockage in the left main coronary...	0	1	None

2 rows × 49 columns

ihiteshmodi

Filtering / Masking in Pandas - Python

If you use the "~" sign while applying the filter / mask, The result will be everything but filtered items

Instead of getting the filtered / masked values you will get all the values excluding the filter / mask

In [11]:
```
#Creating the mask
mask = (df["Name"] == "Jane Doe")

#applying the mask (filtering out the mask by using the "~" symbol)
df.loc[~mask].head(5)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-r
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None
3	William Thompson	Male	62	1	1	1	145	90	100	1	...	Shows a 80% blockage in the left circumflex co...	0	1	None
4	Ashley Johnson	Female	58	1	1	1	135	80	105	1	...	Shows a 90% blockage in the right coronary artery	0	1	None
5	Brian Brown	Male	55	1	1	1	150	95	110	1	...	Shows a 70% blockage in the left anterior	0	1	None

ihiteshmodi

Lets Learn how to use .str method with masking

looking at first 25 unique values in "Cardiac CT" Column as we will apply str method on it

In [26]:
```
df["Cardiac CT"].unique()[0:24]
```

```
array(['Shows a 50% blockage in the left anterior descending coronary artery',
       'Shows a 60% blockage in the right coronary artery',
       'Shows a 75% blockage in the left anterior descending coronary artery',
       'Shows a 80% blockage in the left circumflex coronary artery',
       'Shows a 90% blockage in the right coronary artery',
       'Shows a 70% blockage in the left anterior descending coronary artery',
       'Shows a 70% blockage in the right coronary artery',
       'Shows a 60% blockage in the left main coronary artery',
       'Shows a moderate regurgitation in the mitral valve',
       'Shows a severe regurgitation in the mitral valve',
       'Shows a mild regurgitation in the mitral valve',
       'Shows a moderate stenosis in the mitral valve',
       'Shows a severe stenosis in the mitral valve',
       'Shows a mild stenosis in the mitral valve',
       'Shows a severe stenosis in the aortic valve',
       'Shows a moderate stenosis in the aortic valve',
       'Shows a moderate stenosis in the tricuspid valve',
       'Shows a severe stenosis in the tricuspid valve',
       'Shows a mild stenosis in the tricuspid valve',
       'Shows a moderate stenosis in the pulmonary valve',
       'Shows a mild stenosis in the pulmonary valve',
       'Shows a severe dilatation in the left ventricle',
       'Shows a moderate dilatation in the left ventricle',
       'Shows a mild dilatation in the left ventricle'], dtype=object)
```

Filtering / Masking in Pandas - Python

Lets filter for everythint that contains "blockage" in the "Cardiac CT" Column

In [21]:

```
#Creating the mask
mask = (df["Cardiac CT"].str.contains("blockage"))

#applying the mask (filtering out the mask by using the "~" symbol)
df.loc[mask].head(5)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-r
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None
3	William Thompson	Male	62	1	1	1	145	90	100	1	...	Shows a 80% blockage in the left circumflex co...	0	1	None
4	Ashley Johnson	Female	58	1	1	1	135	80	105	1	...	Shows a 90% blockage in the right coronary	0	1	None

ihiteshmodi

Bonus for staying till the end

Why do we see values 0, 8, 62 etc.. in the index, why cant we see 1,2,3,4,5?

In [23]:
```
#Creating the mask
mask = (df["Name"] == "Jane Doe")

#applying the mask (filtering out the mask by using the "~" symbol)
df.loc[mask].head(5)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-r
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	Non
8	Jane Doe	Female	55	1	1	1	130	80	110	1	...	Shows a 60% blockage in the left main coronary...	0	1	Non
62	Jane Doe	Female	42	1	1	1	120	70	80	1	...	Shows a severe dilatation in the left ventricle	0	1	Non
111	Jane Doe	Female	75	0	1	1	140	90	80	1	...	Shows a 50% dilation of the thoracic aorta	0	1	Non
162	Jane Doe	Female	55	1	0	1	120	80	100	1	...	Shows thickening of the pericardium	0	0	Non

5 rows x 49 columns

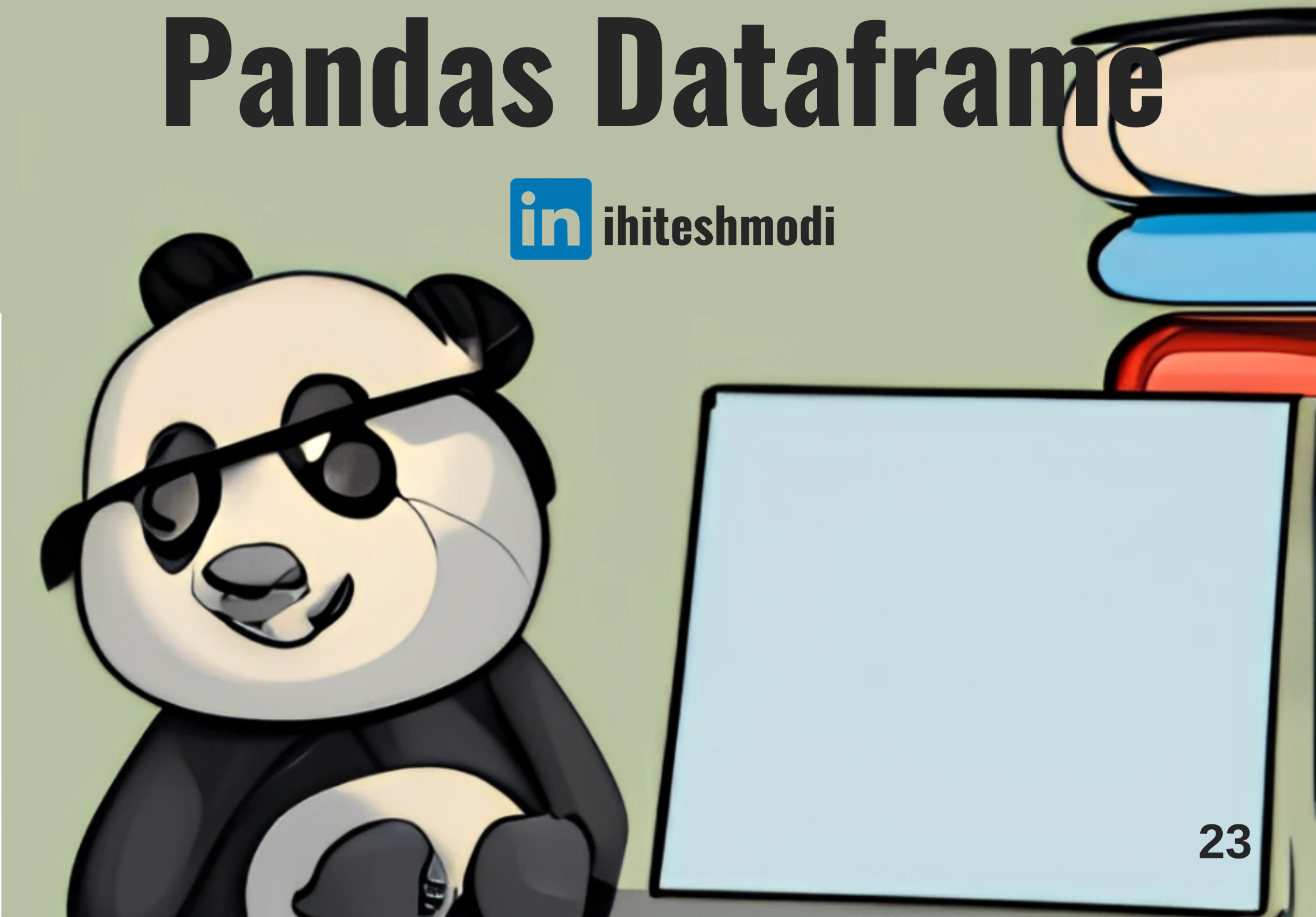
Its because values 1-7 were filtered out so we can see 8th index after 0, Index does not auto reset after filtering / masking the values

ihiteshmodi

Pandas Guide

Updating Column And Row Values in Pandas Dataframe

ihiteshmodi



Updating Column And Row Values in Pandas Dataframe

Today we will learn how to update row and column values in Pandas (Reanaming / Deleting)

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

In [2]:
```
df = pd.read_csv("Heart_disease_details.csv")
```

This is how our dataframe looks like (Heart Disease dataset from kaggle)

In [5]:
```
df.head(5)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-ra
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None
3	William Thompson	Male	62	1	1	1	145	90	100	1	...	Shows a 80% blockage in the left circumflex co...	0	1	None

ihiteshmodi

Updating Column And Row Values in Pandas Dataframe

Get a list of all the columns in our current dataframe

```
In [10]: df.columns
```

```
Index(['Name', 'Gender', 'Age', 'Chest pain', 'Shortness of breath', 'Fatigue',
       'Systolic', 'Diastolic', 'Heart rate (bpm)', 'Lung sounds',
       'Cholesterol level (mg/dL)', 'LDL level (mg/dL)', 'HDL level (mg/dL)',
       'Diabetes', 'Atrial fibrillation', 'Mitral valve prolapse',
       'Rheumatic fever', 'Mitral stenosis', 'Aortic stenosis',
       'Tricuspid stenosis', 'Pulmonary stenosis', 'Dilated cardiomyopathy',
       'Hypertrophic cardiomyopathy', 'Restrictive cardiomyopathy',
       'Arrhythmogenic right ventricular cardiomyopathy',
       'Takotsubo cardiomyopathy', 'Drug use', 'Fever', 'Chills', 'Joint pain',
       'Alcoholism', 'Hypertension', 'Fainting', 'Dizziness', 'Smoking',
       'High cholesterol', 'Echocardiogram', 'Blood culture', 'EKG',
       'Cardiac CT', 'Obesity', 'Murmur', 'Chest x-ray', 'Previous illnesses',
       'Pulmonary function tests', 'Spirometry', 'Diagnosis', 'Medications',
       'Treatment'],
      dtype='object')
```

Renaming the columns

```
In [7]: df2 = df.copy()
```

```
In [17]: df2.columns = ['column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8
```

```
In [18]: df2.columns
```

```
Index(['column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6',
       'column_7', 'column_8', 'column_9', 'column_10', 'column_11',
       'column_12', 'column_13', 'column_14', 'column_15', 'column_16',
       'column_17', 'column_18', 'column_19', 'column_20', 'column_21',
       'column_22', 'column_23', 'column_24', 'column_25', 'column_26',
       'column_27', 'column_28', 'column_29', 'column_30', 'column_31',
       'column_32', 'column_33', 'column_34', 'column_35', 'column_36',
       'column_37', 'column_38', 'column_39', 'column_40', 'column_41',
       'column_42', 'column_43', 'column_44', 'column_45', 'column_46',
       'column_47', 'column_48', 'column_49'],
      dtype='object')
```

Using this method, you can only rename all columns, if you want to rename only few columns, do that using a dictionary (Example on next page)

Updating Column And Row Values in Pandas Dataframe

```
In [35]: df2.rename(columns = {"column_1" : "Col 1", "column_49" : "Col 49"}, inplace = True)

In [36]: df2.columns

Index(['Col 1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6',
       'column_7', 'column_8', 'column_9', 'column_10', 'column_11',
       'column_12', 'column_13', 'column_14', 'column_15', 'column_16',
       'column_17', 'column_18', 'column_19', 'column_20', 'column_21',
       'column_22', 'column_23', 'column_24', 'column_25', 'column_26',
       'column_27', 'column_28', 'column_29', 'column_30', 'column_31',
       'column_32', 'column_33', 'column_34', 'column_35', 'column_36',
       'column_37', 'column_38', 'column_39', 'column_40', 'column_41',
       'column_42', 'column_43', 'column_44', 'column_45', 'column_46',
       'column_47', 'column_48', 'Col 49'],
      dtype='object')
```

Updating Column And Row Values in Pandas Dataframe

Make all the columns uppercase (using str.upper())

In [30]:
```
df.columns = df.columns.str.upper()
```

In [31]:
```
df.columns
```

```
Index(['NAME', 'GENDER', 'AGE', 'CHEST_PAIN', 'SHORTNESS_OF_BREATH', 'FATIGUE',
       'SYSTOLIC', 'DIASTOLIC', 'HEART_RATE_(BPM)', 'LUNG_SOUNDS',
       'CHOLESTEROL_LEVEL_(MG/DL)', 'LDL_LEVEL_(MG/DL)', 'HDL_LEVEL_(MG/DL)',
       'DIABETES', 'ATRIAL_FIBRILLATION', 'MITRAL_VALVE_PROLAPSE',
       'RHEUMATIC_FEVER', 'MITRAL_STENOSIS', 'AORTIC_STENOSIS',
       'TRICUSPID_STENOSIS', 'PULMONARY_STENOSIS', 'DILATED_CARDIOMYOPATHY',
       'HYPERTROPHIC_CARDIOMYOPATHY', 'RESTRICTIVE_CARDIOMYOPATHY',
       'ARRHYTHMOGENIC_RIGHT_VENTRICULAR_CARDIOMYOPATHY',
       'TAKOTSUBO_CARDIOMYOPATHY', 'DRUG_USE', 'FEVER', 'CHILLS', 'JOINT_PAIN',
       'ALCOHOLISM', 'HYPERTENSION', 'FAINTING', 'DIZZINESS', 'SMOKING',
       'HIGH_CHOLESTEROL', 'ECHOCARDIOGRAM', 'BLOOD_CULTURE', 'EKG',
       'CARDIAC_CT', 'OBESITY', 'MURMUR', 'CHEST_X-RAY', 'PREVIOUS_ILLNESSES',
       'PULMONARY_FUNCTION_TESTS', 'SPIROMETRY', 'DIAGNOSIS', 'MEDICATIONS',
       'TREATMENT'],
      dtype='object')
```

Make all the columns lowercase (using str.lower())

In [32]:
```
df.columns = df.columns.str.lower()
```

In [33]:
```
df.columns
```

```
Index(['name', 'gender', 'age', 'chest_pain', 'shortness_of_breath', 'fatigue',
       'systolic', 'diastolic', 'heart_rate_(bpm)', 'lung_sounds',
       'cholesterol_level_(mg/dl)', 'ldl_level_(mg/dl)', 'hdl_level_(mg/dl)',
       'diabetes', 'atrial_fibrillation', 'mitral_valve_prolapse',
       'rheumatic_fever', 'mitral_stenosis', 'aortic_stenosis',
       'tricuspid_stenosis', 'pulmonary_stenosis', 'dilated_cardiomyopathy',
       'hypertrophic_cardiomyopathy', 'restrictive_cardiomyopathy',
       'arrhythmogenic_right_ventricular_cardiomyopathy',
       'takotsubo_cardiomyopathy', 'drug_use', 'fever', 'chills', 'joint_pain',
       'alcoholism', 'hypertension', 'fainting', 'dizziness', 'smoking',
       'high_cholesterol', 'echocardiogram', 'blood_culture', 'ekg',
       'cardiac_ct', 'obesity', 'murmur', 'chest_x-ray', 'previous_illnesses',
       'pulmonary_function_tests', 'spirometry', 'diagnosis', 'medications',
       'treatment'],
      dtype='object')
```

ihiteshmodi

Lets learn how to apply list comprehension on columns now

Make all the columns uppercase (using list comprehension)

In [22]:
```
df.columns
```

```
Index(['NAME', 'GENDER', 'AGE', 'CHEST PAIN', 'SHORTNESS OF BREATH', 'FATIGUE',
       'SYSTOLIC', 'DIASTOLIC', 'HEART RATE (BPM)', 'LUNG SOUNDS',
       'CHOLESTEROL LEVEL (MG/DL)', 'LDL LEVEL (MG/DL)', 'HDL LEVEL (MG/DL)',
       'DIABETES', 'ATRIAL FIBRILLATION', 'MITRAL VALVE PROLAPSE',
       'RHEUMATIC FEVER', 'MITRAL STENOSIS', 'AORTIC STENOSIS',
       'TRICUSPID STENOSIS', 'PULMONARY STENOSIS', 'DILATED CARDIOMYOPATHY',
       'HYPERTROPHIC CARDIOMYOPATHY', 'RESTRICTIVE CARDIOMYOPATHY',
       'ARRHYTHMOGENIC RIGHT VENTRICULAR CARDIOMYOPATHY',
       'TAKOTSUBO CARDIOMYOPATHY', 'DRUG USE', 'FEVER', 'CHILLS', 'JOINT PAIN',
       'ALCOHOLISM', 'HYPERTENSION', 'FAINTING', 'DIZZINESS', 'SMOKING',
       'HIGH CHOLESTEROL', 'ECHOCARDIOGRAM', 'BLOOD CULTURE', 'EKG',
       'CARDIAC CT', 'OBESITY', 'MURMUR', 'CHEST X-RAY', 'PREVIOUS ILLNESSES',
       'PULMONARY FUNCTION TESTS', 'SPIROMETRY', 'DIAGNOSIS', 'MEDICATIONS',
       'TREATMENT'],
      dtype='object')
```

Make all the columns lowercase (using list comprehension)

In [23]:
```
df.columns = [x.lower() for x in df.columns]
```

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

```
Index(['name', 'gender', 'age', 'chest pain', 'shortness of breath', 'fatigue',
       'systolic', 'diastolic', 'heart rate (bpm)', 'lung sounds',
       'cholesterol level (mg/dl)', 'ldl level (mg/dl)', 'hdl level (mg/dl)',
       'diabetes', 'atrial fibrillation', 'mitral valve prolapse',
       'rheumatic fever', 'mitral stenosis', 'aortic stenosis',
       'tricuspid stenosis', 'pulmonary stenosis', 'dilated cardiomyopathy',
       'hypertrophic cardiomyopathy', 'restrictive cardiomyopathy',
       'arrhythmogenic right ventricular cardiomyopathy',
       'takotsubo cardiomyopathy', 'drug use', 'fever', 'chills', 'joint pain',
       'alcoholism', 'hypertension', 'fainting', 'dizziness', 'smoking',
       'high cholesterol', 'echocardiogram', 'blood culture', 'ekg',
       'cardiac ct', 'obesity', 'murmur', 'chest x-ray', 'previous illnesses',
       'pulmonary function tests', 'spirometry', 'diagnosis', 'medications',
       'treatment'],
      dtype='object')
```

Updating Column And Row Values in Pandas Dataframe

Replace spaces in columns with underscore

In [26]:
```
df.columns = df.columns.str.replace(" ", "_")
```

In [27]:
```
df.columns
```

```
Index(['name', 'gender', 'age', 'chest_pain', 'shortness_of_breath', 'fatigue',
       'systolic', 'diastolic', 'heart_rate_(bpm)', 'lung_sounds',
       'cholesterol_level_(mg/dl)', 'ldl_level_(mg/dl)', 'hdl_level_(mg/dl)',
       'diabetes', 'atrial_fibrillation', 'mitral_valve_prolapse',
       'rheumatic_fever', 'mitral_stenosis', 'aortic_stenosis',
       'tricuspid_stenosis', 'pulmonary_stenosis', 'dilated_cardiomyopathy',
       'hypertrophic_cardiomyopathy', 'restrictive_cardiomyopathy',
       'arrhythmogenic_right_ventricular_cardiomyopathy',
       'takotsubo_cardiomyopathy', 'drug_use', 'fever', 'chills', 'joint_pain',
       'alcoholism', 'hypertension', 'fainting', 'dizziness', 'smoking',
       'high_cholesterol', 'echocardiogram', 'blood_culture', 'ekg',
       'cardiac_ct', 'obesity', 'murmur', 'chest_x-ray', 'previous_illnesses',
       'pulmonary_function_tests', 'spirometry', 'diagnosis', 'medications',
       'treatment'],
      dtype='object')
```

Updating Column And Row Values in Pandas Dataframe

Now, Lets see how to make changes to the rows

In [38]:
```
df.head(3)
```

	name	gender	age	chest_pain	shortness_of_breath	fatigue	systolic	diastolic	heart_rate_(bpm)	lung_sounds	...	card
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Show 50% block the le anteri desc.
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Show 60% block the ri coron artery
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Show 75% block the le anteri desc.

3 rows x 49 columns

Lets replace "Jane Doe" in first row with "Hitesh Modi"

We will be using the "Loc" indexers here, if you dont know how to use loc, please refer to my indexers guide

ihiteshmodi

Updating Column And Row Values in Pandas Dataframe

```
In [41]: df.loc[0, "name"]

'Jane Doe'

In [42]: df.loc[0, "name"] = "Hitesh Modi"

In [44]: df.head(3)
```

	name	gender	age	chest_pain	shortness_of_breath	fatigue	systolic	diastolic	heart_rate_(bpm)	lung_sounds	...	card
0	Hitesh Modi	Female	55	1	1	1	140	90	100	1	...	Show 50% block the le anteri desc.
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Show 60% block the ri coron artery
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Show 75% block the le anteri desc.

3 rows x 49 columns

If you do not want to use the index locations, we will have to use the replace method will I will demonstrate in my next guide

ihiteshmodi

Updating Column And Row Values in Pandas Dataframe

Lets see how to update values in multiple columns using indexes

In [48]:
```
df.loc[0, ["gender", "age", "chest_pain"]]
```

```
gender        Female
age               55
chest_pain         1
Name: 0, dtype: object
```

The way to go forwards here is, we inserted multiple columns in the list to get them, now we must insert new values in the same order to change them

In [49]:
```
df.loc[0, ["gender", "age", "chest_pain"]] = ["Male", 24, 0]
```

In [50]:
```
df.head(3)
```

	name	gender	age	chest_pain	shortness_of_breath	fatigue	systolic	diastolic	heart_rate_(bpm)	lung_sounds	...	card
0	Hitesh Modi	Male	24	0	1	1	140	90	100	1	...	Show 50% block the le anteri desc.
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Show 60% block the ri coron artery
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Show 75% block the le anteri desc.

3 rows x 49 columns

ihiteshmodi

Updating Column And Row Values in Pandas Dataframe

If you watched my filtering and masking guide and wonder if we can apply the same ideas here to update values, You can

Lets see how

In [53]:
```
filt = (df["name"] == "Hitesh Modi")
df.loc[filt, ["name", "gender", "age", "chest_pain"]] = ["Jane Doe", "Female", 55, 1]
```

And "Jane Doe" should be back now :)

In [54]:
```
df.head(3)
```

	name	gender	age	ches_pain	shortness_of_breath	fatigue	systolic	diastolic	heart_rate_(bpm)	lung_sounds	...	card
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Show 50% block the le anteri desc.
1	Mark Johnson	[illegible]	[illegible]	1	1	1	150	80	110	1	...	Show 60% block the ri coron artery
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Show 75% block the le anteri desc.

3 rows x 49 columns

This guide involved updating rows based on index numbers which is good to know but has less application in the real world data cleaning scenarios where you would mainly use .map & .replace for editing data in rows, Lets look at them in the next guide, stay tuned and follow me for latest updates

ihiteshmodi

Pandas Guide

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

Lets Learn about .apply, .map, .applymap and .replace methods in pandas today

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

This is how our dataframe looks like, (Heart Disease Dataset from kaggle)

In [2]:
```
df = pd.read_csv("Heart_disease_details.csv")
```

In [4]:
```
df.head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None

3 rows x 49 columns

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

Lets start with learning what is the .apply method, how it works differently on dataframe and series and how to apply functions with it

Starting by observing how .apply method works on dataframe

```
In [6]: df.apply(len)
```

```
Name                                                  334
Gender                                                334
Age                                                   334
Chest pain                                            334
Shortness of breath                                   334
Fatigue                                               334
Systolic                                              334
Diastolic                                             334
Heart rate (bpm)                                      334
Lung sounds                                           334
Cholesterol level (mg/dL)                             334
LDL level (mg/dL)                                     334
HDL level (mg/dL)                                     334
Diabetes                                              334
Atrial fibrillation                                   334
Mitral valve prolapse                                 334
Rheumatic fever                                       334
Mitral stenosis                                       334
Aortic stenosis                                       334
Tricuspid stenosis                                    334
Pulmonary stenosis                                    334
Dilated cardiomyopathy                                334
Hypertrophic cardiomyopathy                           334
Restrictive cardiomyopathy                            334
Arrhythmogenic right ventricular cardiomyopathy       334
Takotsubo cardiomyopathy                              334
Drug use                                              334
Fever                                                 334
Chills                                                334
Joint pain                                            334
Alcoholism                                            334
Hypertension                                          334
Fainting                                              334
Dizziness                                             334
Smoking                                               334
High cholesterol                                      334
```

When we use the len function using .apply method on a dataframe, it gives us number of rows in each column

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

And this is how we can get the minimum value in each Column, For string columns, min is calculated based on placement in the alphabetical order eg: A = 1 and B = 2 so A < B

```
In [14]: df.apply(lambda x: x.min())
```

Name	Adam Lewis
Gender	Female
Age	23
Chest pain	0
Shortness of breath	0
Fatigue	0
Systolic	110
Diastolic	70
Heart rate (bpm)	60
Lung sounds	0
Cholesterol level (mg/dL)	150
LDL level (mg/dL)	95
HDL level (mg/dL)	10
Diabetes	0
Atrial fibrillation	0
Mitral valve prolapse	0
Rheumatic fever	0
Mitral stenosis	0
Aortic stenosis	0
Tricuspid stenosis	0
Pulmonary stenosis	0
Dilated cardiomyopathy	0
Hypertrophic cardiomyopathy	0
Restrictive cardiomyopathy	0
Arrhythmogenic right ventricular cardiomyopathy	0
Takotsubo cardiomyopathy	0
Drug use	0
Fever	0
Chills	0
Joint pain	0
Alcoholism	0
Hypertension	0
Fainting	0
Dizziness	0
Smoking	0

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

But, when we apply to a specific column, it is showing us the length of each string inside the column

```
In [8]: df["Name"].apply(len)
```

```
0      8
1     12
2     11
3     16
4     14
      ..
329   12
330   11
331   10
332   13
333   11
Name: Name, Length: 334, dtype: int64
```

When we use the .apply method on a series, it will be applied on each row of the series one by one

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

We can also use .apply method to apply functions on a row of data

In [9]:
```
def UppercaseFucntion(x):
    return x.upper()
```

In [10]:
```
df["Name"].apply(UppercaseFucntion)
```

```
0            JANE DOE
1         MARK JOHNSON
2          EMILY DAVIS
3      WILLIAM THOMPSON
4        ASHLEY JOHNSON
             ...
329        EMILY WILSON
330         JACOB SMITH
331          JANE SMITH
332       DAVID JOHNSON
333         EMILY SMITH
Name: Name, Length: 334, dtype: object
```

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

We can also Apply Lambda functions using .apply method

```
In [11]: df["Name"].apply(lambda x: x.lower())
```

```
0              jane doe
1         mark johnson
2          emily davis
3      william thompson
4        ashley johnson
              ...
329        emily wilson
330         jacob smith
331          jane smith
332       david johnson
333         emily smith
Name: Name, Length: 334, dtype: object
```

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

Now, Lets learn about .applymap method, PS - This works on entire dataframe only and not on individual series!

for the sake of this tutorial, lets take only the columns with strings

In [29]:
```
df2 = df[["Name", "Gender", "Cardiac CT", "Diagnosis", "Medications", "Treatment"]]
```

In [30]:
```
df2.head(3)
```

	Name	Gender	Cardiac CT	Diagnosis	Medications	Treatment
0	Jane Doe	Female	Shows a 50% blockage in the left anterior desc...	Coronary artery disease (CAD)	Aspirin, metoprolol, atorvastatin	Angioplasty,Coronary artery bypass surgery
1	Mark Johnson	Male	Shows a 60% blockage in the right coronary artery	Coronary artery disease (CAD)	Aspirin, ramipril, atorvastatin	Angioplasty,Coronary artery bypass surgery
2	Emily Davis	Female	Shows a 75% blockage in the left anterior desc...	Coronary artery disease (CAD)	Aspirin, lisinopril, rosuvastatin	Angioplasty,Coronary artery bypass surgery

So now we have a datafarme with string columns only, lets apply string methods now using .applymap and see how this method works on the next page

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

```
In [33]: df2.applymap(str.upper)
```

	Name	Gender	Cardiac CT	Diagnosis	Medications	Treatment
0	JANE DOE	FEMALE	SHOWS A 50% BLOCKAGE IN THE LEFT ANTERIOR DESC...	CORONARY ARTERY DISEASE (CAD)	ASPIRIN, METOPROLOL, ATORVASTATIN	ANGIOPLASTY,CORONARY ARTERY BYPASS SURGERY
1	MARK JOHNSON	MALE	SHOWS A 60% BLOCKAGE IN THE RIGHT CORONARY ARTERY	CORONARY ARTERY DISEASE (CAD)	ASPIRIN, RAMIPRIL, ATORVASTATIN	ANGIOPLASTY,CORONARY ARTERY BYPASS SURGERY
2	EMILY DAVIS	FEMALE	SHOWS A 75% BLOCKAGE IN THE LEFT ANTERIOR DESC...	CORONARY ARTERY DISEASE (CAD)	ASPIRIN, LISINOPRIL, ROSUVASTATIN	ANGIOPLASTY,CORONARY ARTERY BYPASS SURGERY
3	WILLIAM THOMPSON	MALE	SHOWS A 80% BLOCKAGE IN THE LEFT CIRCUMFLEX CO...	CORONARY ARTERY DISEASE (CAD)	ASPIRIN, METOPROLOL, ATORVASTATIN	ANGIOPLASTY,CORONARY ARTERY BYPASS SURGERY
4	ASHLEY JOHNSON	FEMALE	SHOWS A 90% BLOCKAGE IN THE RIGHT CORONARY ARTERY	CORONARY ARTERY DISEASE (CAD)	ASPIRIN, SIMVASTATIN, LISINOPRIL	ANGIOPLASTY,CORONARY ARTERY BYPASS SURGERY
...	...	...	...	...	...	...
329	EMILY WILSON	FEMALE	SHOWS CONSTRICTIVE PERICARDITIS	CONSTRICTIVE PERICARDITIS	IBUPROFEN, COLCHICINE, PREDNISONE	PERICARDIECTOMY
330	JACOB SMITH	MALE	SHOWS CONSTRICTIVE PERICARDITIS	CONSTRICTIVE PERICARDITIS	IBUPROFEN, COLCHICINE, PREDNISONE	PERICARDIECTOMY
331	JANE SMITH	FEMALE	SHOWS CONSTRICTIVE PERICARDITIS	CONSTRICTIVE PERICARDITIS	IBUPROFEN, COLCHICINE, PREDNISONE	PERICARDIECTOMY
332	DAVID JOHNSON	MALE	SHOWS CONSTRICTIVE PERICARDITIS	CONSTRICTIVE PERICARDITIS	IBUPROFEN, COLCHICINE, PREDNISONE	PERICARDIECTOMY
333	EMILY SMITH	FEMALE	SHOWS CONSTRICTIVE PERICARDITIS	CONSTRICTIVE PERICARDITIS	IBUPROFEN, COLCHICINE, PREDNISONE	PERICARDIECTOMY

334 rows x 6 columns

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

As we just saw, All the values became uppercase in dataframe. .applymap method applies a function to EACH AND EVERY SINGLE CELL in the dataframe

In [34]:
```
df2.applymap(str.lower)
```

	Name	Gender	Cardiac CT	Diagnosis	Medications	Treatment
0	jane doe	female	shows a 50% blockage in the left anterior desc...	coronary artery disease (cad)	aspirin, metoprolol, atorvastatin	angioplasty,coronary artery bypass surgery
1	mark johnson	male	shows a 60% blockage in the right coronary artery	coronary artery disease (cad)	aspirin, ramipril, atorvastatin	angioplasty,coronary artery bypass surgery
2	emily davis	female	shows a 75% blockage in the left anterior desc...	coronary artery disease (cad)	aspirin, lisinopril, rosuvastatin	angioplasty,coronary artery bypass surgery
3	william thompson	male	shows a 80% blockage in the left circumflex co...	coronary artery disease (cad)	aspirin, metoprolol, atorvastatin	angioplasty,coronary artery bypass surgery
4	ashley johnson	female	shows a 90% blockage in the right coronary artery	coronary artery disease (cad)	aspirin, simvastatin, lisinopril	angioplasty,coronary artery bypass surgery
...	...	...	...	...	...	...
329	emily wilson	female	shows constrictive pericarditis	constrictive pericarditis	ibuprofen, colchicine, prednisone	pericardiectomy
330	jacob smith	male	shows constrictive pericarditis	constrictive pericarditis	ibuprofen, colchicine, prednisone	pericardiectomy
331	jane smith	female	shows constrictive pericarditis	constrictive pericarditis	ibuprofen, colchicine, prednisone	pericardiectomy
332	david johnson	male	shows constrictive pericarditis	constrictive pericarditis	ibuprofen, colchicine, prednisone	pericardiectomy
333	emily smith	female	shows constrictive pericarditis	constrictive pericarditis	ibuprofen, colchicine, prednisone	pericardiectomy

334 rows × 6 columns

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

Now, Lets have a look at the .map method and .replace method

.map method can be used to replace values, but say if there are a total of 3 values and you use .map method on 2 values, the 2 values will update but the third one will become "nan". .map method discards everything outisde of the values you provide it

In [ ]:

.replace method on the other hand will update the two values and leave the third one as it is

Lets have a look at both the methods with help of an example on the next page starting with .map method

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

```
In [ ]: df2 = df.copy()[0:4] #Preparing a copy of our dataset with only first 4 rows for this tutorial
```

```
In [39]: df2
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-ra
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None
3	William Thompson	Male	62	1	1	1	145	90	100	1	...	Shows a 80% blockage in the left circumflex co...	0	1	None

4 rows x 49 columns

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

We can see that in the "Name" column, we have 4 unique values, "Jane Doe", "Mark Johnson", "Emily Davis" and "William Thompson"

In [ ]:

Lets look at the result we get after using .map method

In [43]:
```
df2["Name"] = df2["Name"].map({"Jane Doe" : "Doe Jane", "Mark Johnson" :  "Johnson Mark"})
```

In [44]:
```
df2
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Doe Jane	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Johnson Mark	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	NaN	Female	0	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None
3	NaN	Male	2	1	1	1	145	90	100	1	...	Shows a 80% blockage in the left circumflex co...	0	1	None

4 rows x 49 columns

We applied the method on first 2 names only so the last two names became null. .map method will disregard everything outside of values that you provide it with

ihiteshmodi

Updating Rows in Pandas using (.apply, .map. applymap & .replace methods)

Lets look at how .replace method would work in this exact same scenario

```
In [45]: df3 = df.copy()[0:4] #Peeparing a copy of our dataset for this tutorial

In [48]: df3["Name"] = df3["Name"].replace({"Jane Doe" : "Doe Jane", "Mark Johnson" :  "Johnson Mark"})
```

```
In [49]: df3
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-ra
0	Doe Jane	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Johnson Mark	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None
3	William Thompson	Male	62	1	1	1	145	90	100	1	...	Shows a 80% blockage in the left circumflex co...	0	1	None

4 rows x 49 columns

The .replace method will update the values we request it to and leave rest of the values as is, it does not disregard the values like .map method

ihiteshmodi

Pandas Guide

IF, Elseif (Elif), Else in Pandas and a 87% faster alternative

ihiteshmodi

IF, Elseif (Elif), Else in Pandas and a 87% faster alternative

Lets learn about how to apply "if else" on pandas dataframe and also learn about a faster alternative to it which also has reproduceability

Lets use the Telecom Churn dataset from Kaggle for this Guide as the dataset has 100K rows and 231 columns

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

In [2]:
```
df = pd.read_csv("Telecom_churn.csv")
```

In [7]:
```
df
```

	mobile_number	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_month_6	last_date_of_month
0	7.000843e+09	109.0	0.0	0.0	0.0	6/30/2014	7/31/2014
1	7.001866e+09	109.0	0.0	0.0	0.0	6/30/2014	7/31/2014
2	7.001626e+09	109.0	0.0	0.0	0.0	6/30/2014	7/31/2014
3	7.001204e+09	109.0	0.0	0.0	0.0	6/30/2014	7/31/2014
4	7.000142e+09	109.0	0.0	0.0	0.0	6/30/2014	7/31/2014
...	...	...	...	...	...	...	...
99994	7.001549e+09	109.0	0.0	0.0	0.0	6/30/2014	7/31/2014
99995	7.000608e+09	109.0	0.0	0.0	0.0	6/30/2014	7/31/2014
99996	7.000088e+09	109.0	0.0	0.0	0.0	6/30/2014	7/31/2014
99997	7.000499e+09	109.0	0.0	0.0	0.0	6/30/2014	7/31/2014
99998	7.001905e+09	109.0	0.0	0.0	0.0	6/30/2014	7/31/2014

99999 rows x 231 columns

ihiteshmodi

IF, Elseif (Elif), Else in Pandas and a 87% faster alternative

Lets have a look at the "Payment Type" column of the dataset and all the unique values in it

In [8]:
```
df["Payment Type"]
```

```
0         NetBanking
1               Cash
2         NetBanking
3         Debit Card
4               Cash
             ...
99994    Credit Card
99995     NetBanking
99996         Wallet
99997           Cash
99998         Wallet
Name: Payment Type, Length: 99999, dtype: object
```

In [ ]:
```
```

In [9]:
```
df["Payment Type"].unique()
```

```
array(['NetBanking', 'Cash', 'Debit Card', 'Credit Card', 'Wallet'],
      dtype=object)
```

ihiteshmodi

IF, Elseif (Elif), Else in Pandas and a 87% faster alternative

We want to create an "If Else" for loop that returns "NB" for Netbanking, "DC" for Debit Card "CC" For Credit Card, "W" for Wallet and "C" for cash, Also we will see how to apply if, elif (short for Else IF) and else in action as well the alternative

```
In [ ]:
```

```
In [ ]:
```

We will put the first 4 fields in an "If Elif" clause and and "Cash" under Else Clause just so we can demonstrate the use of Else clause as well

Lets Create an empty column with Null values where we will put the results of If, Elif, Else there

```
In [10]: df["Payment Type Abbreviations"] = np.NaN
```

ihiteshmodi

IF, Elseif (Elif), Else in Pandas and a 87% faster alternative

Using the Basic "If, Elif, Else" with For loop in pandas

In [12]:

```
%%time
for i in range(len(df)):
    if df["Payment Type"][i] == "Wallet":
        df["Payment Type Abbreviations"][i] = "W"
    elif df["Payment Type"][i] == "Credit Card":
        df["Payment Type Abbreviations"][i] = "CC"
    elif df["Payment Type"][i] == "Debit Card":
        df["Payment Type Abbreviations"][i] = "DC"
    elif df["Payment Type"][i] == "NetBanking":
        df["Payment Type Abbreviations"][i] = "NB"
    else:
        df["Payment Type Abbreviations"][i] = "C" #We are keeping C as an abbreviation for Cash
```

```
CPU times: total: 1.27 s
Wall time: 1.26 s
```

We wil use the %%time magic for measuring how much time it takes for execution of the code

In [ ]:

Took 1.26s to execute on 100K rows

ihiteshmodi

IF, Elseif (Elif), Else in Pandas and a 87% faster alternative

Lets use the Masking method to generate the exact same results as the for loop and check if time it takes to execute the code reduces

In [ ]:

In [14]:
```
%%time
maskw = df["Payment Type"] == "Wallet"
maskcc = df["Payment Type"] == "Credit Card"
maskdc = df["Payment Type"] == "Debit Card"
masknb = df["Payment Type"] == "NetBanking"

df.loc[maskw, "Payment Type Abbreviations"] = "W"
df.loc[maskcc, "Payment Type Abbreviations"] = "CC"
df.loc[maskdc, "Payment Type Abbreviations"] = "DC"
df.loc[masknb, "Payment Type Abbreviations"] = "NB"

#and for else clause, we will use fillna
df["Payment Type Abbreviations"] = df["Payment Type Abbreviations"].fillna("C") # For Cash using for Else
```

```
CPU times: total: 15.6 ms
Wall time: 23 ms
```

Great to see that the time reduced to just 1/6th of the original, this is 83.33% saving in time

This currently is not a reproduceable piece of code, I however will show a reproduceable version of this on the next page

IF, Elseif (Elif), Else in Pandas and a 87% faster alternative

In the reproduceable code, we will use a Dictionary for all of our If, and Elif. The prior value will be the value to check for and the later value will be the result of Then. Eg - "Wallet" : "W" means if our column contains "Wallet" then in our new column return "W"

In [15]:
```
dictionary_of_all_values_to_change = {"Wallet" : "W",
                                      "Credit Card": "CC",
                                      "Debit Card" : "DC",
                                      "NetBanking" : "NB"}
```

In [ ]:
```
'''
For the Arguments piece of this function,
1) df = Dataframe which we will provide
2) searchcolumnname = The column to search in
3) applycolumnname = The column where we will see the results of then clause
4) dictionaryofvalues = the dictionary we created above that has a list of all values
5) elsevalue = the final piece "Else" (can be left blank)
'''
```

In [18]:
```
def fasterforloop(df, searchcolumnname, applycolumnname, dictionaryofvalues, elsevalue = ''):
    for i, j in dictionaryofvalues.items():
        mask = df[searchcolumnname] == i
        df.loc[mask, applycolumnname] == j

    if elsevalue == '':
        pass
    else:
        df = df[applycolumnname].fillna(elsevalue)
```

In [19]:
```
%%time
fasterforloop(df, "Payment Type", "Payment Type Abbreviations", dictionary_of_all_values_to_change, "C")
```

```
CPU times: total: 46.9 ms
Wall time: 26.8 ms
```

ihiteshmodi

We just have to update the dictionary with the values we want and the reproduceabe code will update automatically. and the reproduceable code is also way faster when compared to for loops, Especially when the data is HUGE time savings will be magnanimous

Pandas Guide

Add or remove rows & columns

ihiteshmodi

Add or remove rows & columns

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

```
In [51]: df = pd.read_csv("Heart_disease_details.csv")
```

This is how our dataframe looks like, (Heart Disease Dataset from kaggle)

```
In [52]: df.head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None

3 rows x 49 columns

Lets start with learning how to drop columns as its straight forward ¶

All we have to do is use the drop method using a list

In [53]:
```
df2 = df.copy() # creating a copy dataframe from which we will drop the columns
```

In [55]:
```
df2.drop(columns= ["Name", "Gender", "Age"], inplace = True)
```

In [56]:
```
df2.head(2)
```

	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	Cholesterol level (mg/dL)	LDL level (mg/dL)	HDL level (mg/dL)	...	Cardiac CT	Obesity	Murmur
0	1	1	1	140	90	100	1	220	150	40	...	Shows a 50% blockage in the left anterior desc...	0	1
1	1	1	1	150	80	110	1	210	130	50	...	Shows a 60% blockage in the right coronary artery	0	1

2 rows × 46 columns

And the three columns are dropped

Add or remove rows & columns

When we want to drop rows, We do that by using the index number of the respective row

```
In [57]: df3 = df.copy(2)
```

Say we want to drop index no 1 "Mark Johnson" we do this and the row will be dropped

```
In [58]: df3.drop(index = 1, inplace = True)
```

```
In [59]: df3.head(2)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None

2 rows x 49 columns

We can see that index no 1 is gone and after no 0, we directly have 2 now

Can we also drop multiple indexes? Ofcourse, Using a list of indexes

In [60]:
```
df3.drop(index = [0,2,3], inplace = True)
```

In [61]:
```
df3.head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
4	Ashley Johnson	Female	58	1	1	1	135	80	105	1	...	Shows a 90% blockage in the right coronary artery	0	1	None
5	Brian Brown	Male	55	1	1	1	150	95	110	1	...	Shows a 70% blockage in the left anterior desc...	0	1	None
6	Emily Davis	Female	60	1	1	1	145	90	110	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None

3 rows x 49 columns

Now, our dataset starts from index no 4 indicating rest were dropped :)

But what if we want to drop multiple rows using mask? Why not. Lets see how

Lets run through a basic concept first

In [63]:
```
mask = df3["Age"] >= 50 #so we are filtering for only the rows where age is greter than or equal to 50 years
```

In [64]:
```
df3[mask].shape #after applying the mask, we have 165 rows
```

```
(165, 49)
```

In [ ]:

In [ ]:

In [65]:
```
df3[mask].index
```

```
Int64Index([  4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
            ...
            322, 323, 324, 325, 326, 327, 328, 329, 331, 332],
           dtype='int64', length=165)
```

The moment we call .index method, we get a list of indexes from the dataset where Age is greater than or equal to 55 Years, We can now use these indexes to drop all these rows from the dataset

Add or remove rows & columns

```
In [67]: df3.drop(index = df3[mask].index, inplace = True)
```

```
In [68]: df3.head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Ch x-r
16	Jacob Smith	Male	45	1	1	1	140	80	90	1	...	Shows a severe regurgitation in the mitral valve	0	1	No
17	Emily Davis	Female	32	0	1	1	120	80	90	1	...	Shows a mild regurgitation in the mitral valve	0	1	No
18	Madison Johnson	Female	42	0	1	1	130	80	90	1	...	Shows a moderate regurgitation in the mitral v...	0	1	No

3 rows x 49 columns

Now, our dataset is starting from 16 indicating all the values where Age is less than 50 years have been dropped

What if our index is not basic numbers, instead we have set some string value as our index?

```
In [36]: df4 = df.copy()

In [37]: df4.set_index("Name", inplace = True)
```

As we can see now, "Name" column is our index now

```
In [42]: df4.index
```

```
Index(['Ashley Johnson', 'Brian Brown', 'Emily Davis', 'John Smith',
       'Jane Doe', 'Mark Johnson', 'Jane Smith', 'John Doe', 'Michael Brown',
       'Jessica Davis',
       ...
       'John Smith', 'Jane Doe', 'Emily Johnson', 'John Smith', 'Jane Doe',
       'Emily Wilson', 'Jacob Smith', 'Jane Smith', 'David Johnson',
       'Emily Smith'],
      dtype='object', name='Name', length=330)
```

```
In [39]: df4.head(3###  As we can see now, "Name" column is our index now)
```

Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	Cholesterol level (mg/dL)	...	Cardiac CT	Obesity	Murn
Ashley Johnson	Female	58	1	1	1	135	80	105	1	220	...	Shows a 90% blockage in the right coronary artery	0	1
Brian Brown	Male	55	1	1	1	150	95	110	1	200	...	Shows a 70% blockage in the left anterior desc...	0	1
Emily Davis	Female	60	1	1	1	145	90	110	1	220	...	Shows a 50% blockage in the left anterior desc...	0	1

3 rows x 48 columns

Lets say we want to drop first row, "Ashley Johnson" from the dataset, we do this

In [ ]:

In [43]:
```
df4.drop(index = "Ashley Johnson", inplace = True)
```

In [43]:
```
df4.drop(index = "Ashley Johnson", inplace = True)
```

In [44]:
```
df4.head(3)
```

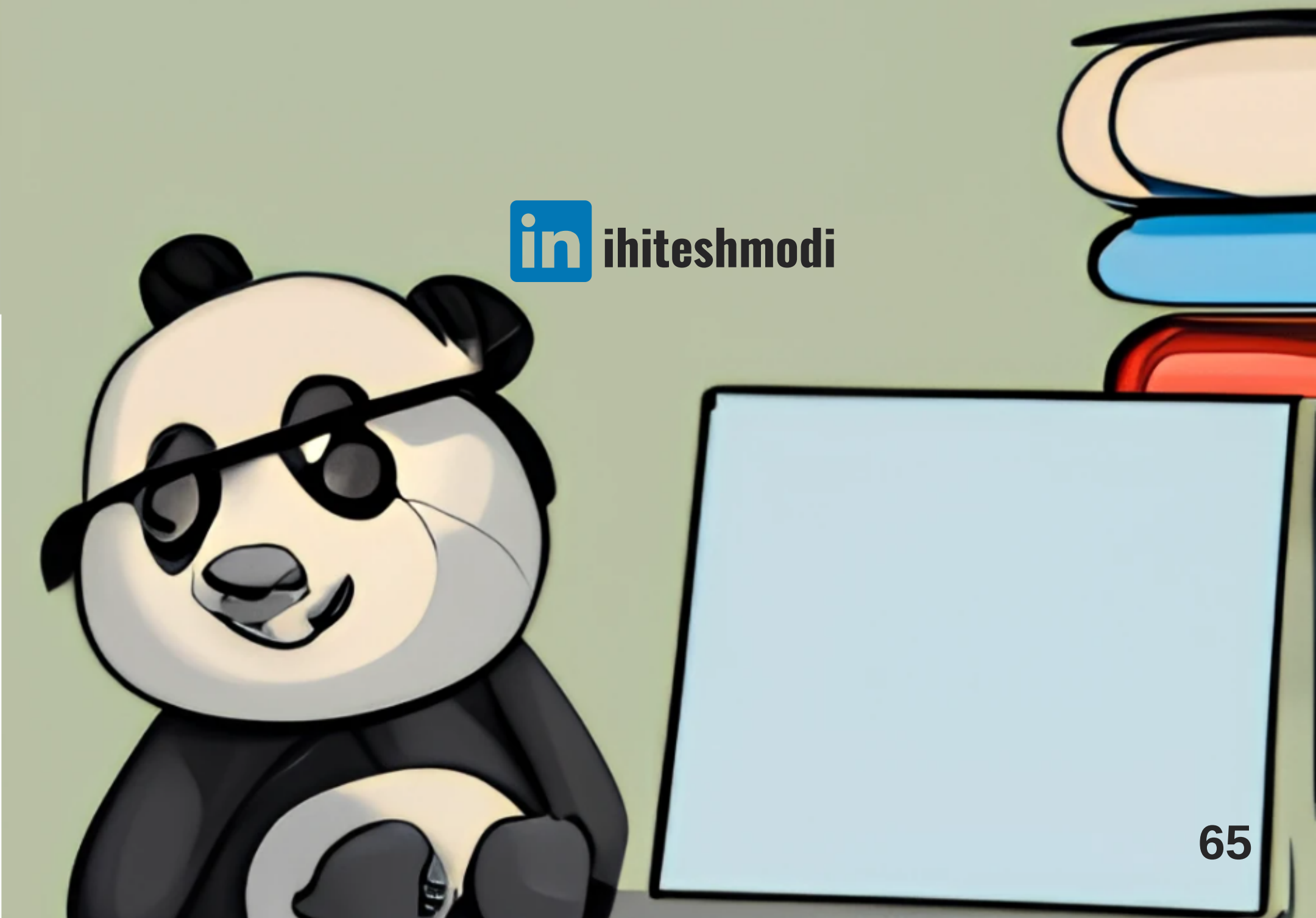
Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	Cholesterol level (mg/dL)	...	Cardiac CT	Obesity	Murmu
Brian Brown	Male	55	1	1	1	150	95	110	1	200	...	Shows a 70% blockage in the left anterior desc...	0	1
Emily Davis	Female	60	1	1	1	145	90	110	1	220	...	Shows a 50% blockage in the left anterior desc...	0	1
John Smith	Male	70	1	1	1	140	90	100	1	240	...	Shows a 70% blockage in the right coronary artery	0	0

3 rows x 48 columns

As we can see, "Ashley Johnson" has been dropped meaning even if we have set some column as our index, we can stil use the values to drop those rows from our dataset

Pandas Guide

Sorting in Pandas



ihiteshmodi

Hi Guys, Lets learn about how to perform Sorting Operations in Pandas

In [ ]:

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

In [2]:
```
df = pd.read_csv("Heart_disease_details.csv")
```

This is how our dataframe looks like, (Heart Disease Dataset from kaggle)

In [3]:
```
df.head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None

3 rows x 49 columns

Lets start by seeing how we can sort the values by "Age" in ascending order

In [4]:
```
df.sort_values(by = "Age").head(5) # No special argument is required for sorting in Ascending Order
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur
246	Olivia Hernandez	Female	23	1	1	1	130	80	110	1	...	None	0	0
94	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild dysfunction of the right ventricle	0	1
109	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild dysfunction of the left ventricle	0	1
234	Jacob Perez	Male	25	1	1	1	130	80	100	1	...	None	0	0
189	Emily Smith	Female	25	1	0	1	120	80	110	1	...	None	0	0

5 rows × 49 columns

Lets start by seeing how we can sort the values by "Age" in Descending order

In [5]:
```
'''
Ascending = False is the argument used for arranging in descending order
'''
df.sort_values(by = "Age", ascending = False).head(5)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Ch x-
138	John Garcia	Male	100	1	1	1	150	90	100	1	...	Shows a 130% dilation of the proximal aorta	0	1	Nor
137	Mary Roberts	Female	95	1	1	1	150	90	100	1	...	Shows a 120% dilation of the proximal aorta	0	1	Nor
136	James Rodriguez	Male	90	1	1	1	150	90	100	1	...	Shows a 110% dilation of the proximal aorta	0	1	Nor
141	David Garcia	Male	85	1	1	1	150	90	100	1	...	Shows a 100% dilation of the proximal aorta	0	1	Nor

Now, Lets sort by Age as well as Heart Rate in ascending order

In [6]:
```
df.sort_values(by = ["Age", "Heart rate (bpm)"]).head(5)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur
246	Olivia Hernandez	Female	23	1	1	1	130	80	110	1	...	None	0	0
217	Emily Davis	Female	25	1	1	1	120	80	90	0	...	None	0	0
73	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild hypertrophy in the left ventricle	0	1
84	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild restriction in the ventricular fi...	0	1
94	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild dysfunction of the right ventricle	0	1

5 rows x 49 columns

We can see that for all 25 years olds, Heart Rate is in Ascending Order

Now, say We want to sort by Age in Ascending order, but by Heart Rate in Descending order, How can we do that? Hint - Using List

In [ ]:
```
'''
Arguments for sorting by are "Age" & "Heart rate (bpm)"

As we want to sort by Age in Ascending and Heart Rate in Descending so arguments will be True and False
(in this order)
'''
```

In [7]:
```
df.sort_values(by = ["Age", "Heart rate (bpm)"], ascending = [True, False]).head(5)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	
246	Olivia Hernandez	Female	23	1	1	1	130	80	110	1	...	None	0	0	Sh no lur str wit ob a..
189	Emily Smith	Female	25	1	0	1	120	80	110	1	...	None	0	0	Nc
200	Olivia Anderson	Female	25	1	0	1	120	80	110	1	...	None	0	0	Nc
240	Emma Williams	Female	25	1	1	1	130	80	110	1	...	None	0	0	Sh no lur str wit ob a..
245	Emily Wilson	Female	25	1	1	1	130	80	110	1	...	None	0	0	Sh no lur str wit ob a..

5 rows x 49 columns

We can now see that for 25 years olds, Heart Rate in Descending Order is starting from 110 instead of 90 (when sorted in Ascending Order)

Now, Say we want to look at values of only 1 column "Name" that too in sorted form, How to?

In [8]:
```
df["Name"].sort_values()
```

```
99          Adam Lewis
216         Adam Smith
126         Adam Smith
267         Adam Smith
33      Alyssa Martinez
            ...
123       Timothy Davis
107     William Johnson
3       William Thompson
312     William Thompson
47      William Thompson
Name: Name, Length: 334, dtype: object
```

The values will now be sorted from A all the way to Z (W in our case as we do not have any names starting X, Y or Z)

To Apply the sorting on our dataframe, We have to use inplace = True

```
In [9]: df.sort_values(by = "Age", inplace = True)
```

```
In [10]: df.head(5) #The dataframe is now sorted
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur
246	Olivia Hernandez	Female	23	1	1	1	130	80	110	1	...	None	0	0
94	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild dysfunction of the right ventricle	0	1
109	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild dysfunction of the left ventricle	0	1
234	Jacob Perez	Male	25	1	1	1	130	80	100	1	...	None	0	0
189	Emily Smith	Female	25	1	0	1	120	80	110	1	...	None	0	0

5 rows x 49 columns

But say if you want to remove the sorting now, How to?

In [11]:
```
df.sort_index(inplace = True)
```

In [12]:
```
df.head(5) #When we sort by index, the dataframe returns to the original format
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Che x-r
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None
3	William Thompson	Male	62	1	1	1	145	90	100	1	...	Shows a 80% blockage in the left circumflex co...	0	1	None
4	Ashley Johnson	Female	58	1	1	1	135	80	105	1	...	Shows a 90% blockage in the right coronary artery	0	1	None

5 rows x 49 columns

Now, Lets look at two methods, nlargest and nsmallest

Say we want to look at top 10 highest Age's in our dataset, This is how we can do that

In [13]:
```
df["Age"].nlargest(10)
```

```
138    100
137     95
136     90
141     85
35      80
117     80
127     80
140     80
308     80
30      75
Name: Age, dtype: int64
```

Great, so these are the top 10 highest ages in our dataset, What if we want to see the entire dataframe for these ages?

```
In [14]: df.nlargest(10, "Age").head(5) #Use the nlargest method
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Ch x-
138	John Garcia	Male	100	1	1	1	150	90	100	1	...	Shows a 130% dilation of the proximal aorta	0	1	Nor
137	Mary Roberts	Female	95	1	1	1	150	90	100	1	...	Shows a 120% dilation of the proximal aorta	0	1	Nor
136	James Rodriguez	Male	90	1	1	1	150	90	100	1	...	Shows a 110% dilation of the proximal aorta	0	1	Nor
141	David Garcia	Male	85	1	1	1	150	90	100	1	...	Shows a 100% dilation of the proximal aorta	0	1	Nor
35	Matthew Turner	Male	80	1	1	1	160	90	70	1	...	Shows a severe stenosis in the aortic valve	0	1	Nor

5 rows × 49 columns

Similarly, If we also wanted to see a sorted dataframe for the 10 lowest Ages, use nsmallest ¶

```
In [15]: df.nsmallest(10, "Age")
```

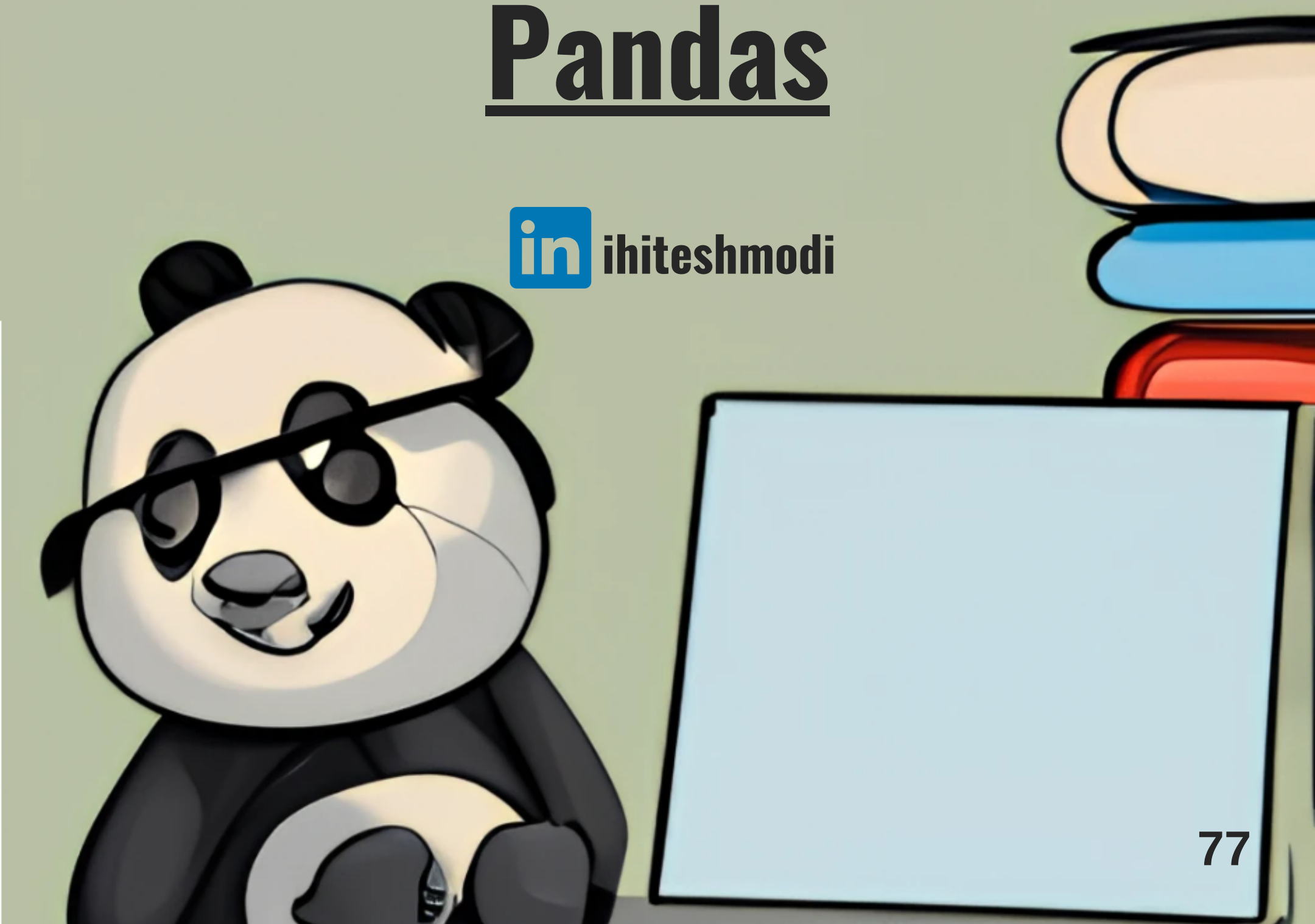
	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur
246	Olivia Hernandez	Female	23	1	1	1	130	80	110	1	...	None	0	0
73	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild hypertrophy in the left ventricle	0	1
84	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild restriction in the ventricular fi...	0	1
94	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild dysfunction of the right ventricle	0	1
109	David Parker	Male	25	0	1	1	130	80	100	0	...	Shows a mild dysfunction of the left ventricle	0	1
169	Emily Turner	Female	25	1	1	1	130	80	100	1	...	Shows pericardial effusion and tamponade	0	0
183	Emily Turner	Female	25	1	1	1	130	80	100	1	...	Shows pericarditis with myocarditis	0	0
189	Emily Smith	Female	25	1	0	1	120	80	110	1	...	None	0	0
200	Olivia Anderson	Female	25	1	0	1	120	80	110	1	...	None	0	0

10 rows × 49 columns

Pandas Guide

Group By and Aggregation in Pandas

ihiteshmodi



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

In [2]:
```
df = pd.read_csv("Heart_disease_details.csv")
```

This is how our dataframe looks like, (Heart Disease Dataset from kaggle)

In [3]:
```
df.head(3)
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray
0	Jane Doe	Female	55	1	1	1	140	90	100	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None
1	Mark Johnson	Male	57	1	1	1	150	80	110	1	...	Shows a 60% blockage in the right coronary artery	0	1	None
2	Emily Davis	Female	60	1	1	1	130	85	95	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None

3 rows x 49 columns

If we scroll to the right, We have the "Previous illnesses" Column

```
In [4]: df.head(3)
```

art ate m)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Chest x-ray	Previous illnesses	Pulmonary function tests	Spirometry	Diagnosis	Medications	Treatment
	1	...	Shows a 50% blockage in the left anterior desc...	0	1	None	None	None	None	Coronary artery disease (CAD)	Aspirin, metoprolol, atorvastatin	Angioplasty,Coronary artery bypass surgery
	1	...	Shows a 60% blockage in the right coronary artery	0	1	None	None	None	None	Coronary artery disease (CAD)	Aspirin, ramipril, atorvastatin	Angioplasty,Coronary artery bypass surgery
	1	...	Shows a 75% blockage in the left anterior desc...	0	1	None	None	None	None	Coronary artery disease (CAD)	Aspirin, lisinopril, rosuvastatin	Angioplasty,Coronary artery bypass surgery

Lets group by "Previous Illnesses" column

```
In [5]: grouped_df = df.groupby(["Previous illnesses"])
```

Now, We can perform basic operations on the groups

```
In [ ]:
```

Lets find out the mean Heart Rate for each of the groups

```
In [6]: grouped_df["Heart rate (bpm)"].mean()
```

```
Previous illnesses
Aortic valve replacement surgery      117.500000
Mitral valve replacement surgery      107.500000
None                                  100.678233
Recent chest surgery                  110.000000
Recent viral infection                103.333333
Tricuspid valve replacement surgery   110.000000
Name: Heart rate (bpm), dtype: float64
```

Lets also look for min and max age in both of the groups

In [7]:
```
grouped_df["Age"].describe()[["min", "max"]]
```

Previous illnesses	min	max
Aortic valve replacement surgery	50.0	70.0
Mitral valve replacement surgery	40.0	65.0
None	23.0	100.0
Recent chest surgery	50.0	60.0
Recent viral infection	35.0	55.0
Tricuspid valve replacement surgery	50.0	55.0

We can also use the agg (aggregate) method

In [8]:
```
grouped_df["Age"].agg(["min", "max"])
```

Previous illnesses	min	max
Aortic valve replacement surgery	50	70
Mitral valve replacement surgery	40	65
None	23	100
Recent chest surgery	50	60
Recent viral infection	35	55
Tricuspid valve replacement surgery	50	55

Lets say we want to look at only the group that underwent recent chest surgery, we have to get only this group from our groups

In [9]:
```
recent_chest_surgery_df = grouped_df.get_group("Recent chest surgery") #use the get group method for this
```

In [10]:
```
recent_chest_surgery_df
```

	Name	Gender	Age	Chest pain	Shortness of breath	Fatigue	Systolic	Diastolic	Heart rate (bpm)	Lung sounds	...	Cardiac CT	Obesity	Murmur	Cl x
158	Emily Brown	Female	50	1	0	1	120	80	110	1	...	Shows thickening of the pericardium	0	0	No
160	David Green	Male	60	1	0	1	130	80	110	1	...	Shows thickening of the pericardium	0	0	No
161	John Smith	Male	60	1	0	1	130	80	110	1	...	Shows thickening of the pericardium	0	0	No
163	Robert Johnson	Male	50	1	0	1	130	80	110	1	...	Shows thickening of the pericardium	0	0	No

4 rows × 49 columns

So, we have only 4 patients that recently underwent chest surgery as we get only 4 rows in return, lets confirm we have the right group using unique values

In [11]:
```
recent_chest_surgery_df["Previous illnesses"].unique()
```

```
array(['Recent chest surgery'], dtype=object)
```

Awesome, having 'Recent chest surgery' as the only unique value confirms that the get groups method worked

What groupbys are really useful for is EDA's

In [12]:
```
grouped_df["Age"].describe()
```

Previous illnesses	count	mean	std	min	25%	50%	75%	max
Aortic valve replacement surgery	4.0	60.000000	8.164966	50.0	57.50	60.0	62.50	70.0
Mitral valve replacement surgery	4.0	51.250000	11.086779	40.0	43.75	50.0	57.50	65.0
None	317.0	49.984227	15.404146	23.0	38.00	48.0	62.00	100.0
Recent chest surgery	4.0	55.000000	5.773503	50.0	50.00	55.0	60.00	60.0
Recent viral infection	3.0	45.000000	10.000000	35.0	40.00	45.0	50.00	55.0
Tricuspid valve replacement surgery	2.0	52.500000	3.535534	50.0	51.25	52.5	53.75	55.0

Here, we can see the basic statistics related to age from all the groups

In [ ]:

In [13]:
```
grouped_df["Age"].describe()[["min", "max"]].sort_values(by = "min")
```

Previous illnesses	min	max
None	23.0	100.0
Recent viral infection	35.0	55.0
Mitral valve replacement surgery	40.0	65.0
Aortic valve replacement surgery	50.0	70.0
Recent chest surgery	50.0	60.0
Tricuspid valve replacement surgery	50.0	55.0

We see that patients who previously underwent surgeries are 40 and above, so maybe we can say that there is a high chance that individuals between the ages of 40 - 70 may undergo atleast 1 surgery?

In [ ]:

This is just one example of how we can use groups for Analysing our datasets but before concluding anything, we may have to look deeper

Lets go deeper into our analysis by grouping by multiple values

In [14]:
```
len(df["Age"].unique())
```

```
34
```

We have a total of 34 unique age values, its ususally a good practice to divide them in age buckets

So lets create an age group column with 4 values, Upto 25 years of age, 26-50, 51-75 and 75+

In [33]:
```
import numpy as np
df["Age Group"] = np.NaN
```

In [34]:
```
#creating the masks

less_than_25 = df["Age"] <= 25
twenty_six_to_50 = (df["Age"] > 25) & (df["Age"] <= 50)
fifty_one_to_75 = (df["Age"] > 50) & (df["Age"] <= 75)
seventy_five_plus = df["Age"] > 75

#applying the masks to create our groupings
df.loc[less_than_25, "Age Group"] = "Upto 25"
df.loc[twenty_six_to_50, "Age Group"] = "26 - 50"
df.loc[fifty_one_to_75, "Age Group"] = "51 - 75"
df.loc[seventy_five_plus, "Age Group"] ="75 +"
```

In [35]:
```
df["Age Group"].unique()
```

```
array(['Upto 25', '26 - 50', '51 - 75', '75 +'], dtype=object)
```

Now, Lets look at how our groups with 2 values look like and what insights we can get from it

In [ ]:

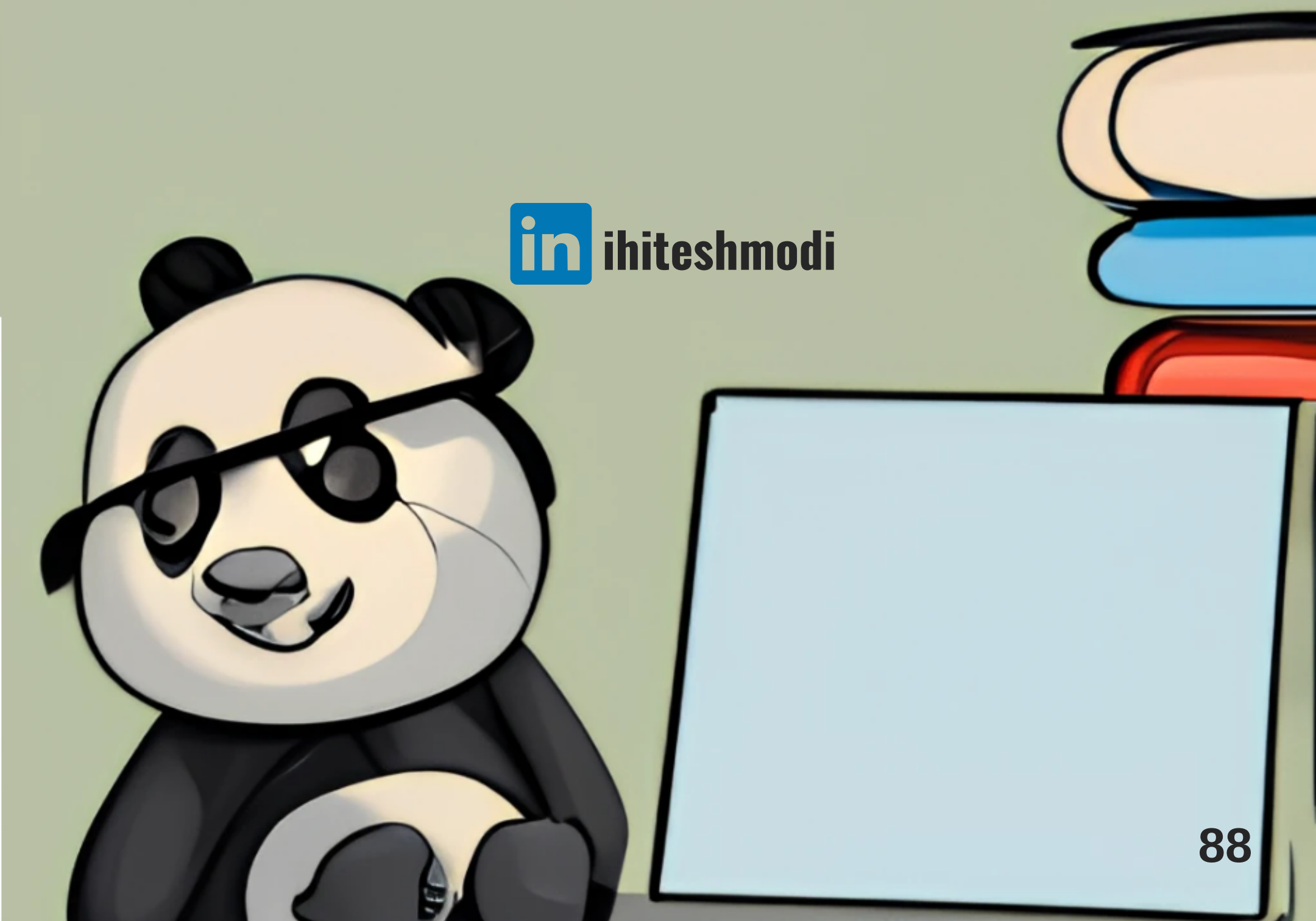
In [59]:
```
new_grouping["Age"].describe()[["count"]].sort_values(by = ["Age Group", "count", "Previous illnesses"])
```

		count
Previous illnesses	Age Group	
Aortic valve replacement surgery	26 - 50	1.0
Tricuspid valve replacement surgery	26 - 50	1.0
Mitral valve replacement surgery	26 - 50	2.0
Recent chest surgery	26 - 50	2.0
Recent viral infection	26 - 50	2.0
None	26 - 50	157.0
Recent viral infection	51 - 75	1.0
Tricuspid valve replacement surgery	51 - 75	1.0
Mitral valve replacement surgery	51 - 75	2.0
Recent chest surgery	51 - 75	2.0
Aortic valve replacement surgery	51 - 75	3.0
None	51 - 75	136.0
	75 +	9.0
	Upto 25	15.0

We can see that majority of the individuals have no previous illnesses and for each age group, there are only a few individuals with previous illnesses, however people below age of 25 have no previous illnesses

Pandas Guide

Handling Null Values



ihiteshmodi

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

Lets custom make our data frame for this tutorial

In [2]:
```
people = {
    "First" : ["Ananya", "Snehal", "Neha", "Komal", "Hitesh", np.nan, None, "NA"],
    "Last" : ["Roy", "Singh", "Jagiasi", "Mandal", "Modi", np.nan, np.nan, "Missing"],
    "Email" : ["r.ananya@fakeemail.com", "s.snehal.com", "j.neha.com", \
               "m.komalfakeemail.com", "m.hiteshfakeemail.com", np.nan, "NA", np.nan]
}
```

In [3]:
```
df = pd.DataFrame(people)
```

In [4]:
```
df2 = df.copy()
```

Handling Null Values

We have a combination of NaN, None and a custom missing values "NA" & "Missing"

In [5]:
```
df
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
5	NaN	NaN	NaN
6	None	NaN	NA
7	NA	Missing	NaN

Drop na by default works in a way where it will drop the rows where even one value is null!

In [6]:
```
df.dropna()
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com

Which is why row number 5,6 and 7 were dropped

Handling Null Values

```
In [7]: # these are the default settings and this is how dropna works when you dont put any arguments

In [8]: df.dropna(axis = "index", how = "any")
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com

Handling Null Values

In [9]:
```
df
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
5	NaN	NaN	NaN
6	None	NaN	NA
7	NA	Missing	NaN

If we want to drop the entire row only when all values are Nulls, We can do this

In [10]:
```
df.dropna(axis = "index", how = "all")
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
6	None	NaN	NA
7	NA	Missing	NaN

Handling Null Values

In [11]:
```
df
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
5	NaN	NaN	NaN
6	None	NaN	NA
7	NA	Missing	NaN

Say if we want to drop the rows only when Last name and Email colums are nulls, we can do this

In [12]:
```
df.dropna(axis = "index", how = "all", subset = ["Last", "Email"])
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
6	None	NaN	NA
7	NA	Missing	NaN

We see row number 7 was not dropped, Why would that be the case?

Handling Null Values

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
6	None	NaN	NA
7	NA	Missing	NaN

Its because NaN and None are python recognised null values

```
In [13]: type(df["Last"][6]) #Numpy Nan's are recognised as floats by pandas
```

```
float
```

"NA" and "Missing" are custom null value and infact are string values not null values

```
In [14]: type(df["Last"][7])
```

```
str
```

ihiteshmodi

We can also check what all values do the dataframe recognise as Nulls

```
In [15]: df.isnull()
```

	First	Last	Email
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False
5	True	True	True
6	True	True	False
7	False	False	True

See, this value is not recognised as a null value by python

```
In [22]: df
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
5	NaN	NaN	NaN
6	None	NaN	NA
7	NaN	Missing	NaN

Handling Null Values

We will have to first convert the values to Null and then try dropna again

In [16]:
```
df["First"].replace("NA", np.NaN, inplace = True)
```

In [17]:
```
df
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
5	NaN	NaN	NaN
6	None	NaN	NA
7	NaN	Missing	NaN

In [18]:
```
df.dropna(axis = "index", how = "all", subset = ["Last", "Email"])
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
6	None	NaN	NA
7	NaN	Missing	NaN

Worked like charm :)

Handling Null Values

If you know all your null values, you can also remove them while reading the csv file

In [19]:
```
na_vals = ["NA", "Missing"]
df_nonna = pd.read_csv("people.csv", na_values = na_vals)
```

In [20]:
```
df2
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
5	NaN	NaN	NaN
6	None	NaN	NA
7	NA	Missing	NaN

In [21]:
```
df_nonna
```

	First	Last	Email
0	Ananya	Roy	r.ananya@fakeemail.com
1	Snehal	Singh	s.snehal.com
2	Neha	Jagiasi	j.neha.com
3	Komal	Mandal	m.komalfakeemail.com
4	Hitesh	Modi	m.hiteshfakeemail.com
5	NaN	NaN	NaN
6	NaN	NaN	NaN
7	NaN	NaN	NaN

See, the values are now already converted to nulls while importing the dataset itself!

ihiteshmodi

Pandas Guide

working with dates and time series data

ihiteshmodi

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

This is how our dataframe looks like

We have 1 hour interval based nifty index data for the past 1 year

```
In [ ]:
```

```
In [2]: df = pd.read_csv("nifty_data.csv")
```

```
In [3]: df
```

	Datetime	Open	High	Low	Close	Adj Close	Volume
0	2022-04-01 09:15:00	17436.900391	17547.099609	17431.699219	17526.750000	17526.750000	0
1	2022-04-01 10:15:00	17526.849609	17556.349609	17517.300781	17548.550781	17548.550781	0
2	2022-04-01 11:15:00	17548.550781	17549.650391	17511.500000	17524.849609	17524.849609	0
3	2022-04-01 12:15:00	17524.650391	17552.050781	17516.300781	17536.599609	17536.599609	0
4	2022-04-01 13:15:00	17537.550781	17593.300781	17537.349609	17583.550781	17583.550781	0
...	...	...	...	...	...	...	...
1732	2023-03-31 11:15:00	17245.099609	17291.949219	17242.099609	17278.699219	17278.699219	0
1733	2023-03-31 12:15:00	17279.000000	17325.199219	17268.650391	17316.650391	17316.650391	0
1734	2023-03-31 13:15:00	17317.500000	17365.550781	17312.400391	17360.800781	17360.800781	0
1735	2023-03-31 14:15:00	17361.750000	17381.300781	17344.150391	17363.199219	17363.199219	0
1736	2023-03-31 15:15:00	17362.650391	17367.500000	17337.699219	17359.750000	17359.750000	0

1737 rows x 7 columns

As we can see below, The date time column is recognised as a object (String), lets start by converting it to correct format first

```
In [4]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1737 entries, 0 to 1736
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Datetime   1737 non-null   object
 1   Open       1737 non-null   float64
 2   High       1737 non-null   float64
 3   Low        1737 non-null   float64
 4   Close      1737 non-null   float64
 5   Adj Close  1737 non-null   float64
 6   Volume     1737 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 95.1+ KB
```

```
In [5]: df["Datetime"] = pd.to_datetime(df["Datetime"])
```

We can now see that the column is being recognised in correct format now

```
In [6]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1737 entries, 0 to 1736
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Datetime   1737 non-null   datetime64[ns]
 1   Open       1737 non-null   float64
 2   High       1737 non-null   float64
 3   Low        1737 non-null   float64
 4   Close      1737 non-null   float64
 5   Adj Close  1737 non-null   float64
 6   Volume     1737 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 95.1 KB
```

Now, we should be able to perfom simple date time related operations on the columns

In [7]:
```
df["Datetime"].max()
```

```
Timestamp('2023-03-31 15:15:00')
```

In [8]:
```
df["Datetime"].min()
```

```
Timestamp('2022-04-01 09:15:00')
```

We can also apply masks on the column now

In [9]:
```
mask = df["Datetime"] >= '2023-01-01'
```

In [10]:
```
df_2023 = df[mask]
```

In [11]:
```
df_2023
```

	Datetime	Open	High	Low	Close	Adj Close	Volume
1303	2023-01-02 09:15:00	18131.699219	18172.949219	18087.550781	18162.900391	18162.900391	0
1304	2023-01-02 10:15:00	18159.400391	18199.300781	18155.400391	18186.500000	18186.500000	0
1305	2023-01-02 11:15:00	18186.099609	18192.949219	18170.500000	18192.050781	18192.050781	0
1306	2023-01-02 12:15:00	18190.650391	18197.300781	18153.900391	18182.349609	18182.349609	0
1307	2023-01-02 13:15:00	18182.250000	18193.449219	18157.949219	18170.099609	18170.099609	0
...	...	...	...	...	...	...	...
1732	2023-03-31 11:15:00	17245.099609	17291.949219	17242.099609	17278.699219	17278.699219	0
1733	2023-03-31 12:15:00	17279.000000	17325.199219	17268.650391	17316.650391	17316.650391	0
1734	2023-03-31 13:15:00	17317.500000	17365.550781	17312.400391	17360.800781	17360.800781	0
1735	2023-03-31 14:15:00	17361.750000	17381.300781	17344.150391	17363.199219	17363.199219	0
1736	2023-03-31 15:15:00	17362.650391	17367.500000	17337.699219	17359.750000	17359.750000	0

434 rows × 7 columns

The max value will still be the same, however the min value should change now

In [ ]:
```
```

In [12]:
```
df_2023["Datetime"].max()
```

```
Timestamp('2023-03-31 15:15:00')
```

In [13]:
```
df_2023["Datetime"].min()
```

```
Timestamp('2023-01-02 09:15:00')
```

We can even set the date column as index and perform loc and slicing operations on it

In [14]:
```
df_2023.set_index("Datetime", inplace = True)
```

In [15]:
```
df_2023
```

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-02 09:15:00	18131.699219	18172.949219	18087.550781	18162.900391	18162.900391	0
2023-01-02 10:15:00	18159.400391	18199.300781	18155.400391	18186.500000	18186.500000	0
2023-01-02 11:15:00	18186.099609	18192.949219	18170.500000	18192.050781	18192.050781	0
2023-01-02 12:15:00	18190.650391	18197.300781	18153.900391	18182.349609	18182.349609	0
2023-01-02 13:15:00	18182.250000	18193.449219	18157.949219	18170.099609	18170.099609	0
...	...	...	...	...	...	...
2023-03-31 11:15:00	17245.099609	17291.949219	17242.099609	17278.699219	17278.699219	0
2023-03-31 12:15:00	17279.000000	17325.199219	17268.650391	17316.650391	17316.650391	0
2023-03-31 13:15:00	17317.500000	17365.550781	17312.400391	17360.800781	17360.800781	0
2023-03-31 14:15:00	17361.750000	17381.300781	17344.150391	17363.199219	17363.199219	0
2023-03-31 15:15:00	17362.650391	17367.500000	17337.699219	17359.750000	17359.750000	0

434 rows x 6 columns

In [ ]:
```
'''
Note, the year has to be passed as a sting, if you pass it as numbers
it wont work as the column is datetime format and not int or float
'''
```

In [17]:
```
df_2023.loc["2023"]
```

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-02 09:15:00	18131.699219	18172.949219	18087.550781	18162.900391	18162.900391	0
2023-01-02 10:15:00	18159.400391	18199.300781	18155.400391	18186.500000	18186.500000	0
2023-01-02 11:15:00	18186.099609	18192.949219	18170.500000	18192.050781	18192.050781	0
2023-01-02 12:15:00	18190.650391	18197.300781	18153.900391	18182.349609	18182.349609	0
2023-01-02 13:15:00	18182.250000	18193.449219	18157.949219	18170.099609	18170.099609	0
...	...	...	...	...	...	...
2023-03-31 11:15:00	17245.099609	17291.949219	17242.099609	17278.699219	17278.699219	0
2023-03-31 12:15:00	17279.000000	17325.199219	17268.650391	17316.650391	17316.650391	0
2023-03-31 13:15:00	17317.500000	17365.550781	17312.400391	17360.800781	17360.800781	0
2023-03-31 14:15:00	17361.750000	17381.300781	17344.150391	17363.199219	17363.199219	0
2023-03-31 15:15:00	17362.650391	17367.500000	17337.699219	17359.750000	17359.750000	0

434 rows x 6 columns

We can also filter for a specific date range using slicing

In [18]:
```
df_2023['2023-01-01':'2023-01-15']
```

Datetime	Open	High	Low	Close	Adj Close	Volume
2023-01-02 09:15:00	18131.699219	18172.949219	18087.550781	18162.900391	18162.900391	0
2023-01-02 10:15:00	18159.400391	18199.300781	18155.400391	18186.500000	18186.500000	0
2023-01-02 11:15:00	18186.099609	18192.949219	18170.500000	18192.050781	18192.050781	0
2023-01-02 12:15:00	18190.650391	18197.300781	18153.900391	18182.349609	18182.349609	0
2023-01-02 13:15:00	18182.250000	18193.449219	18157.949219	18170.099609	18170.099609	0
...	...	...	...	...	...	...
2023-01-13 11:15:00	17830.349609	17853.099609	17813.699219	17851.099609	17851.099609	0
2023-01-13 12:15:00	17851.150391	17965.050781	17849.349609	17965.050781	17965.050781	0
2023-01-13 13:15:00	17964.349609	17998.949219	17945.150391	17989.750000	17989.750000	0
2023-01-13 14:15:00	17988.199219	17992.500000	17934.550781	17957.800781	17957.800781	0
2023-01-13 15:15:00	17958.099609	17965.699219	17947.000000	17950.500000	17950.500000	0

70 rows x 6 columns

Lets also perform basic statistics

In [19]:
```
df_2023['2023-01-01':'2023-01-15']["Close"].mean()
```

```
18002.51704799107
```

In [20]:
```
df_2023['2023-01-01':'2023-01-15']["High"].max()
```

```
18251.849609375
```

In [21]:
```
df_2023['2023-01-01':'2023-01-15']["Low"].min()
```

```
17762.25
```

Though our datframe is on hourly level and if we remove timstamps the data will loose its value, lets make some changes to date formats for the sake of this tutorial

In [ ]:

In [22]:
```
df
```

	Datetime	Open	High	Low	Close	Adj Close	Volume
0	2022-04-01 09:15:00	17436.900391	17547.099609	17431.699219	17526.750000	17526.750000	0
1	2022-04-01 10:15:00	17526.849609	17556.349609	17517.300781	17548.550781	17548.550781	0
2	2022-04-01 11:15:00	17548.550781	17549.650391	17511.500000	17524.849609	17524.849609	0
3	2022-04-01 12:15:00	17524.650391	17552.050781	17516.300781	17536.599609	17536.599609	0
4	2022-04-01 13:15:00	17537.550781	17593.300781	17537.349609	17583.550781	17583.550781	0
...	...	...	...	...	...	...	...
1732	2023-03-31 11:15:00	17245.099609	17291.949219	17242.099609	17278.699219	17278.699219	0
1733	2023-03-31 12:15:00	17279.000000	17325.199219	17268.650391	17316.650391	17316.650391	0
1734	2023-03-31 13:15:00	17317.500000	17365.550781	17312.400391	17360.800781	17360.800781	0
1735	2023-03-31 14:15:00	17361.750000	17381.300781	17344.150391	17363.199219	17363.199219	0
1736	2023-03-31 15:15:00	17362.650391	17367.500000	17337.699219	17359.750000	17359.750000	0

1737 rows × 7 columns

Lets remove the timestamp from date column (all we do is exclude timestamps in the code)

In [23]:
```
'''
Smallcase "y" instead of capital "Y" will return just last 2 digits of they year instead of all 4 digits
'''
df["Datetime"] = pd.to_datetime(df["Datetime"]).dt.strftime('%Y-%m-%d')
```

In [24]:
```
df
```

	Datetime	Open	High	Low	Close	Adj Close	Volume
0	2022-04-01	17436.900391	17547.099609	17431.699219	17526.750000	17526.750000	0
1	2022-04-01	17526.849609	17556.349609	17517.300781	17548.550781	17548.550781	0
2	2022-04-01	17548.550781	17549.650391	17511.500000	17524.849609	17524.849609	0
3	2022-04-01	17524.650391	17552.050781	17516.300781	17536.599609	17536.599609	0
4	2022-04-01	17537.550781	17593.300781	17537.349609	17583.550781	17583.550781	0
...	...	...	...	...	...	...	...
1732	2023-03-31	17245.099609	17291.949219	17242.099609	17278.699219	17278.699219	0
1733	2023-03-31	17279.000000	17325.199219	17268.650391	17316.650391	17316.650391	0
1734	2023-03-31	17317.500000	17365.550781	17312.400391	17360.800781	17360.800781	0
1735	2023-03-31	17361.750000	17381.300781	17344.150391	17363.199219	17363.199219	0
1736	2023-03-31	17362.650391	17367.500000	17337.699219	17359.750000	17359.750000	0

1737 rows x 7 columns

As we can see now, we have dates only and timestamps are gone

What if we want to format dates in dd-mm-yyyy format?

In [ ]:

In [25]:
```
df["Datetime"] = pd.to_datetime(df["Datetime"]).dt.strftime('%d-%m-%Y')
```

In [26]:
```
df
```

	Datetime	Open	High	Low	Close	Adj Close	Volume
0	01-04-2022	17436.900391	17547.099609	17431.699219	17526.750000	17526.750000	0
1	01-04-2022	17526.849609	17556.349609	17517.300781	17548.550781	17548.550781	0
2	01-04-2022	17548.550781	17549.650391	17511.500000	17524.849609	17524.849609	0
3	01-04-2022	17524.650391	17552.050781	17516.300781	17536.599609	17536.599609	0
4	01-04-2022	17537.550781	17593.300781	17537.349609	17583.550781	17583.550781	0
...	...	...	...	...	...	...	...
1732	31-03-2023	17245.099609	17291.949219	17242.099609	17278.699219	17278.699219	0
1733	31-03-2023	17279.000000	17325.199219	17268.650391	17316.650391	17316.650391	0
1734	31-03-2023	17317.500000	17365.550781	17312.400391	17360.800781	17360.800781	0
1735	31-03-2023	17361.750000	17381.300781	17344.150391	17363.199219	17363.199219	0
1736	31-03-2023	17362.650391	17367.500000	17337.699219	17359.750000	17359.750000	0

1737 rows x 7 columns

This was basic overview of how to deal with date types, Dates however is an extensive topic not only in pandas but in all of the analytics related tools, the best way to learn about more about them is going through the documentation

I will attach some more strftime codes below which I got from geeks for greeks

ihiteshmodi

format code	meaning	example
%a	Abbreviated weekday name	Sun, Mon
%A	Full weekday name	Sunday, Monday
%w	Weekday as decimal number	0...6
%d	Day of the month as a zero-padded decimal	01, 02
%-d	day of the month as decimal number	1, 2..
%b	Abbreviated month name	Jan, Feb
%m	month as a zero padded decimal number	01, 02
%-m	month as a decimal number	1, 2

Source - https://www.geeksforgeeks.org/python-datetime-strptime-function/

ihiteshmodi

%B	Full month name	January, February
%y	year without century as a zero padded decimal number	99, 00
%-y	year without century as a decimal number	0, 99
%Y	year with century as a decimal number	2000, 1999
%H	hour(24 hour clock) as a zero padded decimal number	01, 23
%-H	hour(24 hour clock) as a decimal number	1, 23
%I	hour(12 hour clock) as a zero padded decimal number	01, 12
%-I	hour(12 hour clock) as a decimal number	1, 12
%p	locale's AM or PM	AM, PM
%M	Minute as a zero padded decimal number	01, 59
%-M	Minute as a decimal number	1, 59

Source - https://www.geeksforgeeks.org/python-datetime-strptime-function/

ihiteshmodi

%S	Second as a zero padded decimal number	01, 59
%-S	Second as a decimal number	1, 59
%f	microsecond as a decimal number, zero padded on the left side	000000, 999999
%z	UTC offset in the form +HHMM or -HHMM	
%Z	Time zone name	
%j	day of the year as a zero padded decimal number	001, 365
%-j	day of the year as a decimal number	1, 365
%U	Week number of the year (Sunday being the first)	0, 6
%W	Week number of the year	00, 53
%c	locale's appropriate date and time representation	Mon Sep 30 07:06:05 2013

Source - https://www.geeksforgeeks.org/python-datetime-strptime-function/

ihiteshmodi

%x	locale's appropriate date representation	11/30/98
%X	locale's appropriate time representation	10:03:43
%%	A literal '%' character	%

Source - https://www.geeksforgeeks.org/python-datetime-strptime-function/