Class Examples Week 3

Dr. M Nouman Durrani

Data Cleaning

- Data scientists spend a large amount of their time cleaning datasets and getting them down to a form with which they can work.
- Data scientists argue that the initial steps of obtaining and cleaning data constitute 80% of the job.

Python Libraries For Data Science

- Python comes with numerous libraries for scientific computing, analysis, visualization etc. Some of them are listed below:
  1. **Numpy** – NumPy is a core library of Python for Data Science
  - It is used for scientific computing, which contains a powerful n-dimensional array object and provide tools for integrating C, C++ etc.
  - It can also be used as multi-dimensional container for generic data where you can perform various Numpy Operations and special functions

Python Libraries For Data Science

2. Matplotlib – Matplotlib is a powerful library for visualization in Python.

- It can be used in Python scripts, shell, web application servers and other GUI toolkits
- You can use different types of plots

3. Scikit-learn – Scikit learn is one of the main attractions, where you can implement machine learning using Python.

- It contains simple and efficient tools for data analysis and mining purposes.
- You can implement various algorithm, such as logistic regression, time series algorithm using scikit-learn.

4. Seaborn – Seaborn is a statistical plotting library in Python

- Seaborn has a high level interface to draw statistical graphics

5. Pandas – Pandas is an important library in Python for data science.

- It is used for data manipulation and analysis.
- It is well suited for different data such as tabular, ordered and unordered time series, matrix data etc.

Common operations for data manipulation

Pandas – Pandas is an important library in Python for data science.

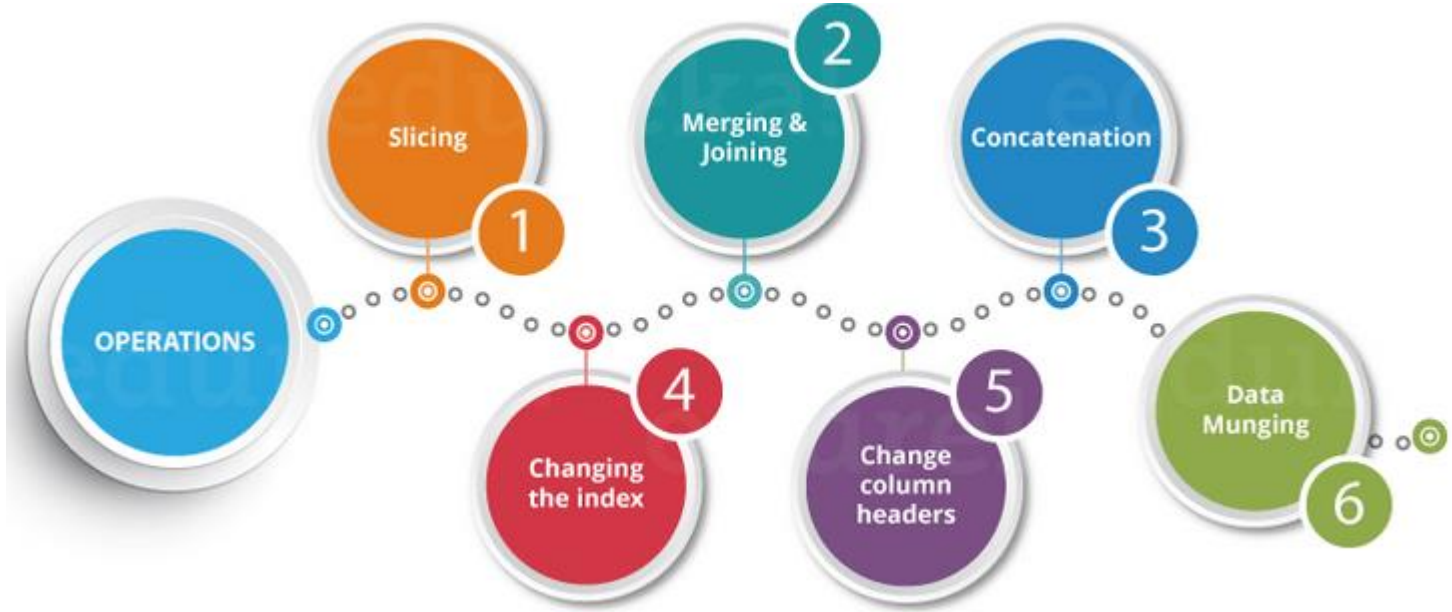
It is used for data manipulation and analysis.

It is well suited for different data such as tabular, ordered and unordered time series, matrix data etc.

Data frame is a 2-dimensional data structure and a most common pandas object. So first, let's create a data frame.

```
import pandas as pd

XYZ_web= {'Day':[1,2,3,4,5,6],
"Visitors":[1000,
700,6000,1000,400,350],
"Bounce_Rate":[20,20, 23,15,10,34]}

df= pd.DataFrame(XYZ_web)

print(df)
```

The code above will convert a dictionary into a pandas Data Frame along with index to the left.



Output:

		Bounce_Rate	Day	Visitors
1	0	20	1	1000
2	1	20	2	700
3	2	23	3	6000
4	3	15	4	1000
5	4	10	5	400
6	5	34	6	350

	Bounce rate	Day	Visitors
0	20	1	1000
1	20	2	700
2	23	3	6000
3	15	4	1000
4	10	5	400
5	34	6	350

Slicing the starting 2 rows

	Bounce rate	Day	Visitors
0	20	1	1000
1	20	2	700

Slicing the t 2 rows

	Bounce rate	Day	Visitors
4	10	5	400
5	34	6	350

print(df.head(2))

Output:

```
   Bounce_Rate Day Visitors
0           20   1     1000
1           20   2      700
```

print(df.tail(2))

```
   Bounce_Rate Day Visitors
4           10   5      400
5           34   6      350
```

Different ways to create a dataframe

Creating Pandas DataFrame from lists of lists.

```
# Import pandas library
import pandas as pd

# initialize list of lists
data = [['tom', 10], ['nick', 15], ['juli', 14]]

# Create the pandas DataFrame
df = pd.DataFrame(data, columns = ['Name', 'Age'])

# print dataframe.
df
```

	Name	Age
0	tom	10
1	nick	15
2	juli	14

Method #2: Creating DataFrame from dict of narray/lists

To create DataFrame from dict of narray/list, all the narray must be of same length.

If index is passed then the length index should be equal to the length of arrays.

If no index is passed, then by default, index will be range(n) where n is the array length.

```
import pandas as pd


# initialize data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'], 'Age':[20, 21, 19, 18]}
 # Create DataFrame
df = pd.DataFrame(data)


# Print the output.
df
```

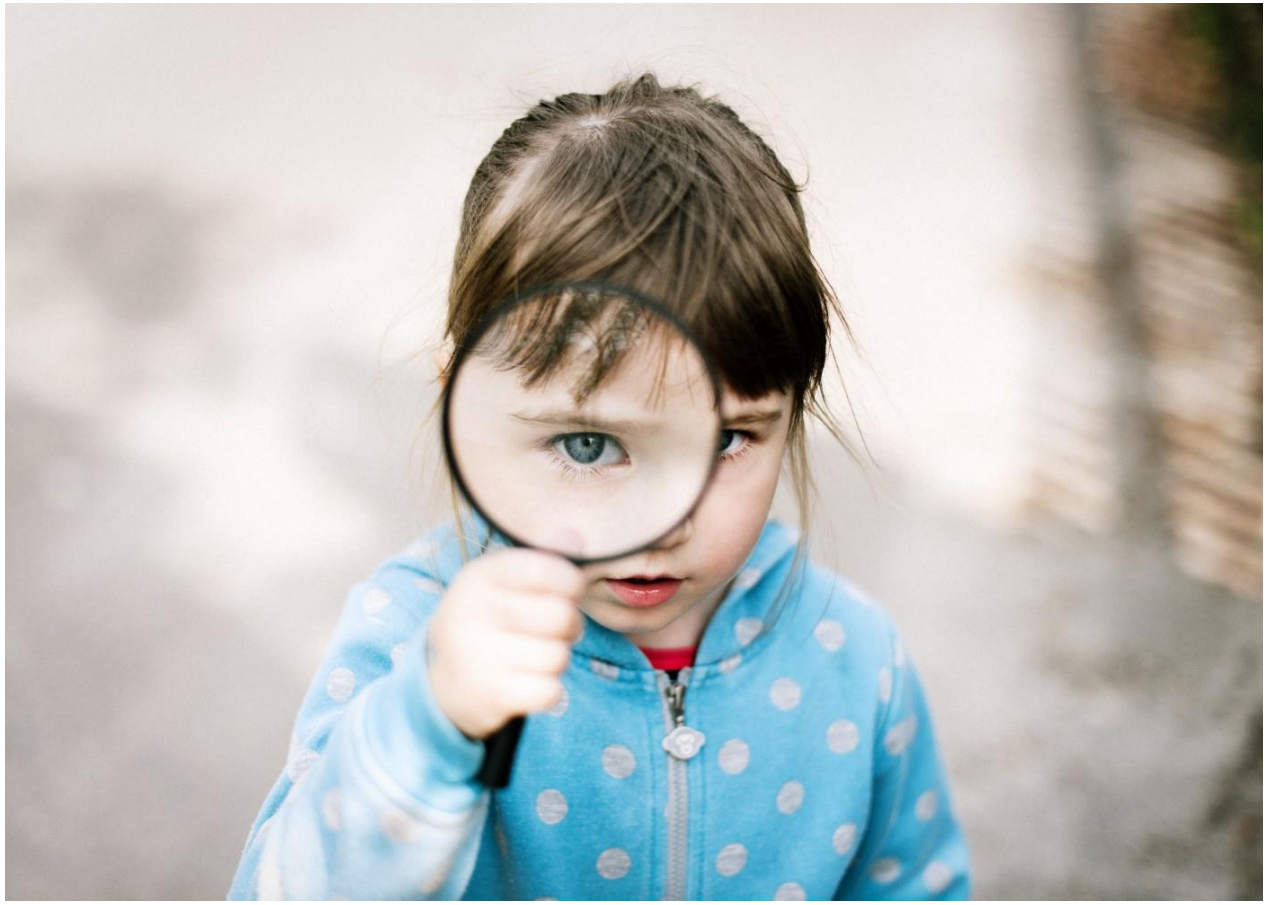
	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

Method #3: Creates a indexes DataFrame using arrays

```
# DataFrame using arrays.
import pandas as pd


# initialize data of lists.
data = {'Name':['Tom', 'Jack', 'nick', 'juli'],
        'marks':[99, 98, 95, 90]}


# Creates pandas DataFrame.
df = pd.DataFrame(data, index =['rank1',
                                'rank2',
                                'rank3',
                                'rank4'])


# print the data
df
```

	Name	marks
rank1	Tom	99
rank2	Jack	98
rank3	nick	95
rank4	juli	90

Handling Missing Values



Data Cleaning

- **Data cleaning** can be a tedious task.
- It's the start of a new project and you're excited to apply some machine learning models.
- You take a look at the data and quickly realize it's an **absolute mess**.
- According to IBM Data Analytics you can expect to spend up to **80% of your time cleaning data**

Missing Source

- **Sources of Missing Values**
- Why data is missing:
  - User forgot to fill in a field.
  - Data was lost while transferring manually from a legacy database.
  - There was a programming error.
  - Users chose not to fill out a field tied to their beliefs about how the results would be used or interpreted.
  - simple random mistakes

Missing Source

- It’s important to understand these different types of missing data from a statistics point of view.

- The type of missing data will influence how you deal with filling in the missing values.

Example Data

ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

Visualizing with Head Function

- Look of Some Observation

```
# Importing libraries
import pandas as pd
import numpy as np

# Read csv file into a pandas dataframe
df = pd.read_csv("property data.csv")

# Take a look at the first few rows
print df.head()


Out:
   ST_NUM    ST_NAME OWN_OCCUPIED  NUM_BEDROOMS
0   104.0     PUTNAM            Y           3.0
1   197.0  LEXINGTON            N           3.0
2     NaN  LEXINGTON            N           3.0
3   201.0   BERKELEY          NaN           1.0
4   203.0   BERKELEY            Y           3.0
```

Data Cleaning

Features

- `ST_NUM` : Street number
- `ST_NAME` : Street name
- `OWN_OCCUPIED` : Is the residence owner occupied
- `NUM_BEDROOMS` : Number of bedrooms

Types of Features

- Expected Types
  - `ST_NUM` : float or int… some sort of numeric type
  - `ST_NAME` : string
  - `OWN_OCCUPIED` : string… Y (“Yes”) or N (“No”)
  - `NUM_BEDROOMS` : float or int, a numeric type

Types of Missing Values

- **Standard Missing Values**
  - **The missing values that Pandas Can detect**

ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

- Pandas will recognize both

empty cells and “NA” types as missing values

```
# Looking at the ST_NUM column
print df['ST_NUM']
print df['ST_NUM'].isnull()


Out:
0    104.0
1    197.0
2      NaN
3    201.0
4    203.0
5    207.0
6      NaN
7    213.0
8    215.0

Out:
0    False
1    False
2     True
3    False
4    False
5    False
6     True
7    False
8    False
```

Handling Missing Values

- **Non-Standard Missing Val**
  - **Pandas Can not Detect**

ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIE
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

```
# Looking at the NUM_BEDROOMS column
print df['NUM_BEDROOMS']
print df['NUM_BEDROOMS'].isnull()


Out:
0      3
1      3
2    n/a
3      1
4      3
5    NaN
6      2
7     --
8     na

Out:
0    False
1    False
2    False
3    False
4    False
5     True
6    False
7    False
8    False
```

Handling Missing Values

- Handling non-standard missing values

```
# Making a list of missing value types
missing_values = ["n/a", "na", "--"]
df = pd.read_csv("property data.csv", na_values = missing_values)
```

```
# Looking at the NUM_BEDROOMS column
print df['NUM_BEDROOMS']
print df['NUM_BEDROOMS'].isnull()
```

```
Out:
0    3.0
1    3.0
2    NaN
3    1.0
4    3.0
5    NaN
6    2.0
7    NaN
8    NaN

Out:
0    False
1    False
2     True
3    False
4    False
5     True
6    False
7     True
8     True
```

Unexpected Missing Values

- **Unexpected Missing Values**

ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

```
# Looking at the OWN_OCCUPIED column
print df['OWN_OCCUPIED']
print df['OWN_OCCUPIED'].isnull()

# Looking at the ST_NUM column
Out:
0       Y
1       N
2       N
3      12
4       Y
5       Y
6     NaN
7       Y
8       Y

Out:
0    False
1    False
2    False
3    False
4    False
5    False
6     True
7    False
8    False
```

Handling unexpected-missing values

1. Loop through the OWN_OCCUPIED column
2. Try and turn the entry into an integer
3. If the entry can be changed into an integer, enter a missing value
4. If the number can't be an integer, we know it's a string, so keep going

```
# Detecting numbers
cnt=0
for row in df['OWN_OCCUPIED']:
    try:
        int(row)
        df.loc[cnt, 'OWN_OCCUPIED']=np.nan
    except ValueError:
        pass
    cnt+=1
```

Summarization

- **Summarizing Missing Values**

```
# Total missing values for each feature
print df.isnull().sum()


Out:
ST_NUM          2
ST_NAME         0
OWN_OCCUPIED    2
NUM_BEDROOMS    4

 # Any missing values?
 print df.isnull().values.any()


 Out:
 True

  # Total number of missing values
  print df.isnull().sum().sum()


  Out:
  8
```

Replacing Missing Values

- **Replacing**

```
# Replace missing values with a number
df['ST_NUM'].fillna(125, inplace=True)
```

```
# Location based replacement
df.loc[2,'ST_NUM'] = 125
```

```
# Replace using median
median = df['NUM_BEDROOMS'].median()
df['NUM_BEDROOMS'].fillna(median, inplace=True)
```

Merging & Joining

	HPI	IND_GDP	Int_Rate
0	80	50	2
1	90	45	1
2	70	45	2
3	60	67	3

- In merging, you can merge two data frames to form a single data frame.
- Create two data frames, which has some key-value pairs and then merge the data frames together.

```
import pandas as pd

df1= pd.DataFrame({
"HPI":[80,90,70,60],"Int_Rate":[2,1,2,3],"IND_GDP":[50,45,45,67]}, index=[2001,
2002,2003,2004])

df2=pd.DataFrame({ "HPI":[80,90,70,60],"Int_Rate":[2,1,2,3],"IND_GDP":[50,45,45,67]},
index=[2005, 2006,2007,2008])

merged= pd.merge(df1,df2)

print(merged)
```

Let “HPI” column to be common and for everything else, I want separate columns.

```
df1 = pd.DataFrame({"HPI":[80,90,70,60],"Int_Rate":[2,1,2,3],
"IND_GDP":[50,45,45,67]}, index=[2001, 2002,2003,2004])

df2 =
pd.DataFrame({"HPI":[80,90,70,60],"Int_Rate":[2,1,2,3],"IND_GDP":[50,45,45,67]},
index=[2005, 2006,2007,2008])

merged= pd.merge(df1,df2,on ="HPI")

print(merged)
```

	HPI	Int_Rate_x	IND_GDP_x	Int_Rate_y	IND_GDP_y
0	80	2	50	2	50
1	90	1	45	1	45
2	70	2	45	2	45
3	60	3	67	3	67

Joining two dataframes

```
df1 = pd.DataFrame({"Int_Rate":[2,1,2,3], "IND_GDP":[50,45,45,67]}, index=[2001,
2002,2003,2004])

df2 = pd.DataFrame({"Low_Tier_HPI":[50,45,67,34],"Unemployment":[1,3,5,6]},
index=[2001, 2003,2004,2004])

joined= df1.join(df2)
print(joined)
```

Output:

```
      IND_GDP  Int_Rate  Low_Tier_HPI  Unemployment
2001       50         2          50.0           1.0
2002       45         1           NaN           NaN
2003       45         2          45.0           3.0
2004       67         3          67.0           5.0
2004       67         3          34.0           6.0
```

Concatenation

- Concatenation basically glues the dataframes together
- You can select the dimension on which you want to concatenate
- Use “pd.concat” and pass in the list of dataframes to concatenate together

```
df1 = pd.DataFrame({"HPI":[80,90,70,60],"Int_Rate":[2,1,2,3],
"IND_GDP":[50,45,45,67]}, index=[2001, 2002,2003,2004])

df2 =
pd.DataFrame({"HPI":[80,90,70,60],"Int_Rate":[2,1,2,3],"IND_GDP":[50,45,45,67
]}, index=[2005, 2006,2007,2008])

concat= pd.concat([df1,df2])

print(concat)
```

	HPI	IND_GDP	Int_Rate
2001	80	50	2
2002	90	45	1
2003	70	45	2
2004	60	67	3
2005	80	50	2
2006	90	45	1
2007	70	45	2
2008	60	67	3

Specify axis=1 in order to join, merge or concatenate along the columns.

```
df1 = pd.DataFrame({"HPI":[80,90,70,60],"Int_Rate":[2,1,2,3],
"IND_GDP":[50,45,45,67]}, index=[2001, 2002,2003,2004])

df2 =
pd.DataFrame({"HPI":[80,90,70,60],"Int_Rate":[2,1,2,3],"IND_GDP":[50,45,4
5,67]}, index=[2005, 2006,2007,2008])

concat= pd.concat([df1,df2],axis=1)

print(concat)
```

```
      HPI  IND_GDP  Int_Rate HPI  IND_GDP Int_Rate
2001  80.0  50.0        2.0   NaN    NaN     NaN
2002  90.0  45.0        1.0   NaN    NaN     NaN
2003  70.0  45.0        2.0   NaN    NaN     NaN
2004  60.0  67.0        3.0   NaN    NaN     NaN
2005  NaN   NaN         NaN   80.0   50.0    2.0
2006  NaN   NaN         NaN   90.0   45.0    1.0
2007  NaN   NaN         NaN   70.0   45.0    2.0
2008  NaN   NaN         NaN   60.0   67.0    3.0
```

Change the index

- How to change the index values in a dataframe
- For example, let us create a dataframe with some key value pairs in a dictionary and change the index values
- Consider the example below:

```
import pandas as pd

df= pd.DataFrame({"Day":[1,2,3,4], "Visitors":[200,
100,230,300], "Bounce_Rate":[20,45,60,10]})

df.set_index("Day", inplace= True)

print(df)
```

```
      Bounce_Rate  Visitors
Day
1              20       200
2              45       100
3              60       230
4              10       300
```

Change the Column Headers

- Let us take the same example, where we will change the column header from “Visitors” to “Users”

```
import pandas as pd

df = pd.DataFrame({"Day":[1,2,3,4], "Visitors":[200,
100,230,300], "Bounce_Rate":[20,45,60,10]})

df = df.rename(columns={"Visitors":"Users"})

print(df)
```

```
   Bounce_Rate  Day  Users
0           20    1    200
1           45    2    100
2           60    3    230
3           10    4    300
```

Data Munging

```
import pandas as pd

country= pd.read_csv("heart.csv",index_col=0)

country.to_html('edu.html')
```

- In Data munging, you can convert a particular data into a different format.
- For example, if you have a .csv file, you can convert it into .html or any other data format as well

	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
age													
63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
49	1	1	130	266	0	1	171	0	0.6	2	0	2	1

Example Use Case 2

Dropping Columns in a DataFrame

- First, let’s create a DataFrame out of the CSV file ‘BL-Flickr-Images-Book.csv’.

```
import pandas as pd
import numpy as np
df = pd.read_csv('BL-Flickr-Images-Book.csv')
df.head()
```

Pandas provides a drop() function to remove unwanted columns or rows from a DataFrame.

Dropping Columns in a DataFrame

```
to_drop = ['Edition Statement', 'Corporate Author', 'Corporate Contributors', 'Former owner',
        'Engraver', 'Contributors', 'Issuance type', 'Shelfmarks']
```

```
df.drop(to_drop, inplace=True, axis=1)
```

Or

```
df.drop(columns=to_drop, inplace=True)
```

We call the drop() function on our object, passing in the inplace parameter as True and the axis parameter as 1.

This tells Pandas that we want the changes to be made directly in our object and that it should look for the values to be dropped in the columns of the object.

Setting Index in a DataFrame

Let’s replace the existing index with this column using set_index:

```
df = df.set_index('Identifier')
df.head()
```

We can access each record in a straightforward way with loc[].

```
df.loc[206]
```

Setting Index in a DataFrame

- df = df.set_index(...). This is because, by default, the method returns a modified copy of our object and does not make the changes directly to the object.

- We can avoid this by setting the inplace parameter:

```
df.set_index('Identifier', inplace=True)
```

When **inplace** = **True** , the data is modified **in place**, which means it will return nothing and the dataframe is now updated. When **inplace** = False , which is the default, then the operation is performed and it returns a copy of the object. You then need to save it to something.

Example Use Case 3

Data Preparation Steps:

- Predict the occurrence of diabetes making use of the entire lifecycle

Step 1:

- Collect the data based on the medical history of the patient

We have the various attributes as mentioned below.

Attributes:

npreg – Number of times pregnant
glucose – Plasma glucose concentration
bp – Blood pressure
skin – Triceps skinfold thickness
bmi – Body mass index
ped – Diabetes pedigree function
age – Age
income – Income

;npreg;glu;bp;skin;bmi;ped;age,income
1;6;148;72;35;33.6;0.627;50
2;1;85;66;29;26.6;0.351;31
3;1;89;80;23;28.1;0.167;21
4;3;78;50;32;31;0.248;26
5;2;197;70;45;30.5;0.158;53
6;5;166;72;19;25.8;0.587;51
7;0;118;84;47;45.8;0.551;31
8;1;103;30;38;43.3;0.183;33
9;3;126;88;41;39.3;0.704;27
10;9;119;80;35;29;0.263;29
11;1;97;66;15;23.2;0.487;22
12;5;109;75;26;36;0.546;60
13;3;88;58;11;24.8;0.267;22
14;10;122;78;31;27.6;0.512;45
15;4;97;60;33;24;0.966;33
16;9;102;76;37;32.9;0.665;46
17;2;90;68;42;38.2;0.503;27
18;4;111;72;47;37.1;1.39;56
19;3;180;64;25;34;0.271;26
20;7;106;92;18;39;0.235;48
21;9;171;110;24;45.4;0.721;54

Step 2:

- We need to clean and prepare the data for data analysis.
- This data has a lot of inconsistencies like:
  - missing values,
  - blank columns,
  - abrupt values and
  - incorrect data format

which need to be cleaned.

	npreg	glu	bp	skin	bmi	ped	age	income
1	6	148	72	35	33.6	0.627	50	
2	1	85	66	29	26.6	0.351	31	
3	1	89	6600	23	28.1	0.167	21	
4	3	78	50	32	31	0.248	26	
5	2	197	70	45	30.5	0.158	53	
6	5	166	72	19	25.8	0.587	51	
7	0	118	84	47	45.8	0.551	31	
8	one	103	30	38	43.3	0.183	33	
9	3	126	88	41	39.3	0.704	27	
10	9	119	80	35	29	0.263	29	
11	1	97	66	15	23.2	0.487	22	
12	5	109	75	26	36	0.546	60	
13	3	88	58	11	24.8	0.267	22	
14	10	122	78	31	27.6	0.512	45	
15	4		60	33	24	0.966	33	
16	9	102	76	37	32.9	0.665	46	
17	2	90	68	42	38.2	0.503	27	
18	4	111	72	47	37.1	1.39	56	
19	3	180	64	25	34	0.271	26	
20	7	106	92	18		0.235	48	
21	9	171	110	24	45.4	0.721	54	

Step 2

- In the column *npreg*, “one” is written in words, whereas it should be in the numeric form like 1
- In column *bp* one of the values is 6600 which is impossible (at least for humans) as bp cannot go up to such huge value
- The *Income* column is blank and also makes no sense in predicting diabetes.
  - Therefore, it is redundant to have it here and should be removed from the table.
- We will clean and preprocess this data by:
  - removing the outliers
  - filling up the null values and
  - normalizing the data type

	npreg	glu	bp	skin	bmi	ped	age	income
1	6	148	72	35	33.6	0.627	50	
2	1	85	66	29	26.6	0.351	31	
3	1	89	6600	23	28.1	0.167	21	
4	3	78	50	32	31	0.248	26	
5	2	197	70	45	30.5	0.158	53	
6	5	166	72	19	25.8	0.587	51	
7	0	118	84	47	45.8	0.551	31	
8	one	103	30	38	43.3	0.183	33	
9	3	126	88	41	39.3	0.704	27	
10	9	119	80	35	29	0.263	29	
11	1	97	66	15	23.2	0.487	22	
12	5	109	75	26	36	0.546	60	
13	3	88	58	11	24.8	0.267	22	
14	10	122	78	31	27.6	0.512	45	
15	4		60	33	24	0.966	33	
16	9	102	76	37	32.9	0.665	46	
17	2	90	68	42	38.2	0.503	27	
18	4	111	72	47	37.1	1.39	56	
19	3	180	64	25	34	0.271	26	
20	7	106	92	18		0.235	48	
21	9	171	110	24	45.4	0.721	54	

	npreg	glu	bp	skin	bmi	ped	age
1	6	148	72	35	33.6	0.627	50
2	1	85	66	29	26.6	0.351	31
3	1	89	80	23	28.1	0.167	21
4	3	78	50	32	31	0.248	26
5	2	197	70	45	30.5	0.158	53
6	5	166	72	19	25.8	0.587	51
7	0	118	84	47	45.8	0.551	31
8	1	103	30	38	43.3	0.183	33
9	3	126	88	41	39.3	0.704	27
10	9	119	80	35	29	0.263	29
11	1	97	66	15	23.2	0.487	22
12	5	109	75	26	36	0.546	60
13	3	88	58	11	24.8	0.267	22
14	10	122	78	31	27.6	0.512	45
15	4	97	60	33	24	0.966	33
16	9	102	76	37	32.9	0.665	46
17	2	90	68	42	38.2	0.503	27
18	4	111	72	47	37.1	1.39	56
19	3	180	64	25	34	0.271	26
20	7	106	92	18	39	0.235	48
21	9	171	110	24	45.4	0.721	54

Data Entry Errors

Value	Count
Good	1598647
Bad	1354468
Godo	15
Bade	1

- Human or machine
- Missing values and typo errors
- Transmission error
- Extraction error
- Transformation error
- Solution: Generate frequency tables of variables.

```
if x == "Godo":
x = "Good"
if x == "Bade":
x = "Bad"
```

Data Preparation:

```
# Drop the columns where all elements are missing values:
df.dropna(axis=1, how='all')

# Drop the columns where any of the elements are missing values
df.dropna(axis=1, how='any')

# Keep only the rows which contain 2 missing values maximum
df.dropna(thresh=2)
```

Data Preparation:

```
# Fill all missing values with the mean of the particular column
df.fillna(df.mean())

# Fill any missing value in column 'A' with the column median
df['A'].fillna(df['A'].median())

# Fill any missing value in column 'Depeche' with the column mode
df['Depeche'].fillna(df['Depeche'].mode())
```

Data Preparation:

- #df is the data frame name where csv data is loaded.
- #drop id and use inplace = True only when you want to modifiy original dataframe, otherwise you can make a copy and use it.
- df.drop(['Number'],axis=1, inplace=True)

- #Removing column data
- df.drop(columns=['City'], inplace=True)

- #Remove rows which contains null values.
- df_1 = df.dropna(subset =['Gender','Age']

Data Preparation:

- #Filling with mean value
- df['Income'] = df['Income'].fillna((df['Income'].mean()))

- #Filling with median value
- df['Age'] = df['Age'].fillna((df['Age'].median()))

- #Filling with mode value
- df['Age'] = df['Age'].fillna((df['Age'].mode()))