Data Analysis

Data analysis is a process of **inspecting, cleansing, transforming,** and **modelling data** with the goal of discovering useful information, informing conclusions, and supporting decision-making.

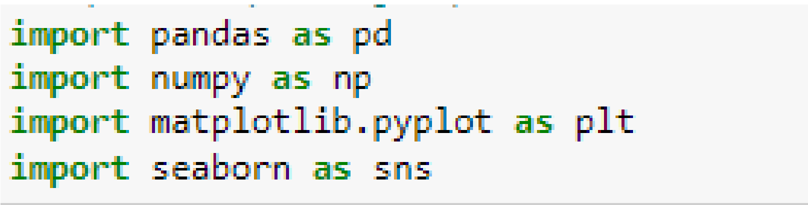
EDA (Exploratory Data Analysis)

- Understanding the data we have
- investigating the dataset to discover patterns, and anomalies (outliers), Null values. form hypotheses based on our understanding of the dataset
- Generating summary statistics for numerical data
- graphical representations to understand the data better

Steps Of EDA

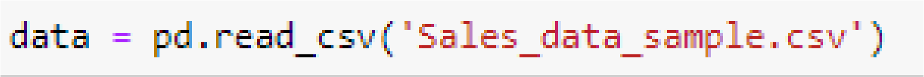
(Using Python**)**

1. Import important Libraries



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

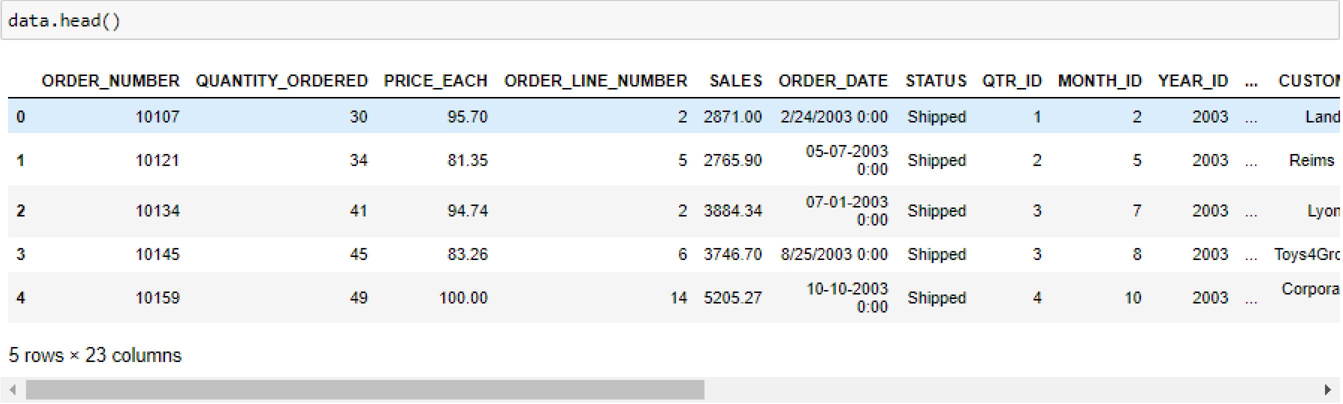
2. Import the dataset



```
data = pd.read_csv('Sales_data_sample.csv')
```

'Sales_data_sample.csv' is name of the file

3. Validate few lines of dataset

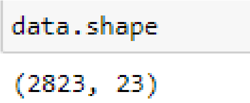


```
data.head()
```

	ORDER_NUMBER	QUANTITY_ORDERED	PRICE_EACH	ORDER_LINE_NUMBER	SALES	ORDER_DATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	CUSTOI
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	Land
1	10121	34	81.35	5	2765.90	05-07-2003 0:00	Shipped	2	5	2003	...	Reims
2	10134	41	94.74	2	3884.34	07-01-2003 0:00	Shipped	3	7	2003	...	Lyon
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	Toys4Grc
4	10159	49	100.00	14	5205.27	10-10-2003 0:00	Shipped	4	10	2003	...	Corpora

5 rows × 23 columns

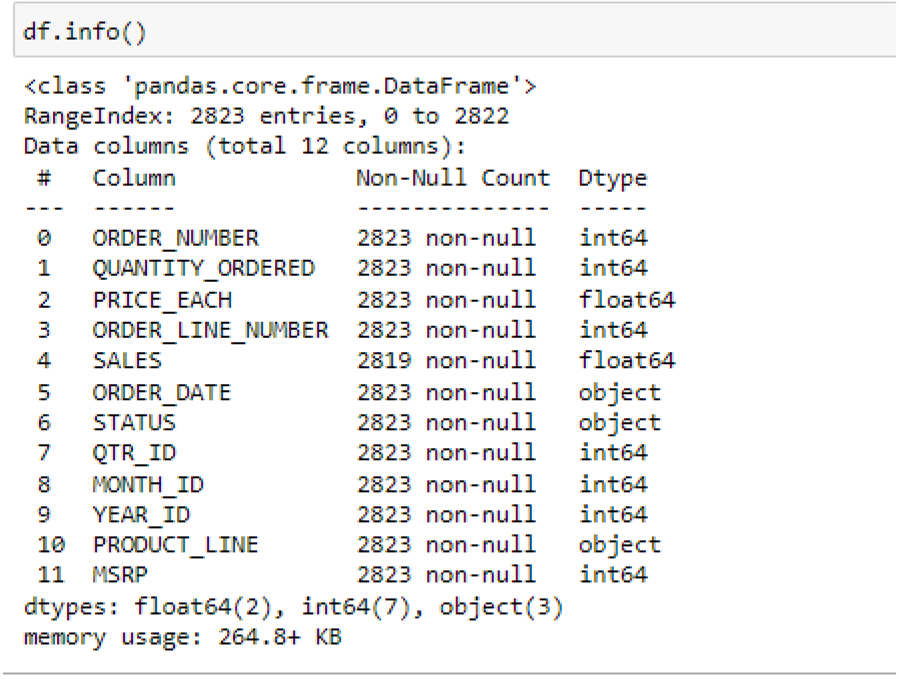
4. Check the shape of data (Rows, Columns)



```
data.shape
```

```
(2823, 23)
```

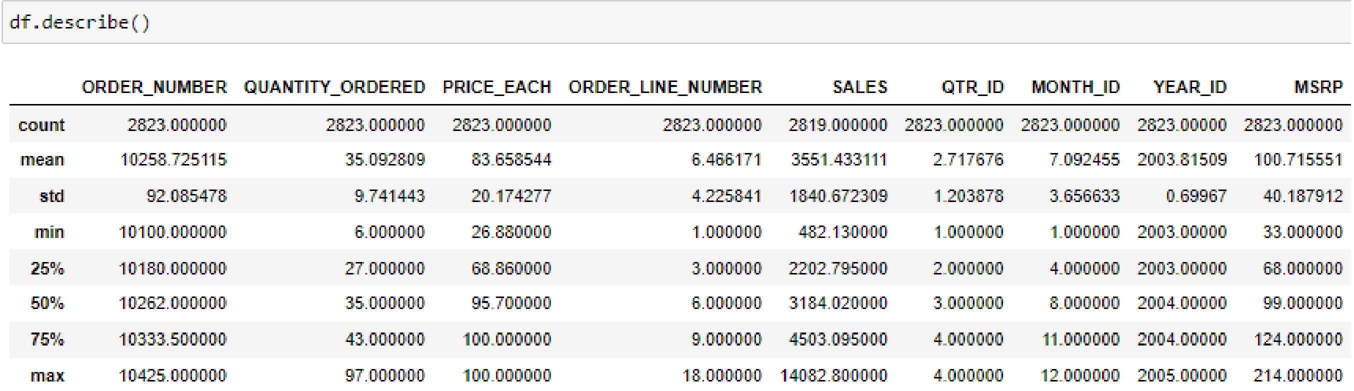
5. Check for data type of columns to verify the data format



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ORDER_NUMBER      2823 non-null   int64
 1   QUANTITY_ORDERED  2823 non-null   int64
 2   PRICE_EACH        2823 non-null   float64
 3   ORDER_LINE_NUMBER 2823 non-null   int64
 4   SALES             2819 non-null   float64
 5   ORDER_DATE        2823 non-null   object
 6   STATUS            2823 non-null   object
 7   QTR_ID            2823 non-null   int64
 8   MONTH_ID          2823 non-null   int64
 9   YEAR_ID           2823 non-null   int64
 10  PRODUCT_LINE      2823 non-null   object
 11  MSRP              2823 non-null   int64
dtypes: float64(2), int64(7), object(3)
memory usage: 264.8+ KB
```

6. Check for description of data (Count, Mean, std, min, max, 25%, 50% etc.)

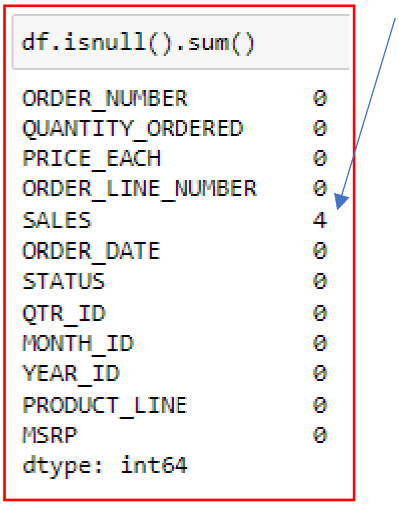


```
df.describe()
```

	ORDER_NUMBER	QUANTITY_ORDERED	PRICE_EACH	ORDER_LINE_NUMBER	SALES	QTR_ID	MONTH_ID	YEAR_ID	MSRP
count	2823.000000	2823.000000	2823.000000	2823.000000	2819.000000	2823.000000	2823.000000	2823.00000	2823.000000
mean	10258.725115	35.092809	83.658544	6.466171	3551.433111	2.717676	7.092455	2003.81509	100.715551
std	92.085478	9.741443	20.174277	4.225841	1840.672309	1.203878	3.656633	0.69967	40.187912
min	10100.000000	6.000000	26.880000	1.000000	482.130000	1.000000	1.000000	2003.00000	33.000000
25%	10180.000000	27.000000	68.860000	3.000000	2202.795000	2.000000	4.000000	2003.00000	68.000000
50%	10262.000000	35.000000	95.700000	6.000000	3184.020000	3.000000	8.000000	2004.00000	99.000000
75%	10333.500000	43.000000	100.000000	9.000000	4503.095000	4.000000	11.000000	2004.00000	124.000000
max	10425.000000	97.000000	100.000000	18.000000	14082.800000	4.000000	12.000000	2005.00000	214.000000

7. Missing values
   a. Check for missing values
      i. Values can be Null, Nan
      ii. Can be an object (? #, *)
   b. Treat the missing Values

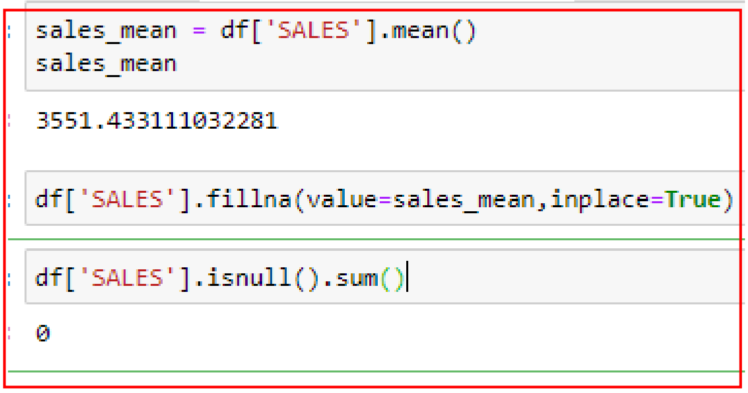
4 null values for Sales



```
df.isnull().sum()
```

```
ORDER_NUMBER         0
QUANTITY_ORDERED     0
PRICE_EACH           0
ORDER_LINE_NUMBER    0
SALES                4
ORDER_DATE           0
STATUS               0
QTR_ID               0
MONTH_ID             0
YEAR_ID              0
PRODUCT_LINE         0
MSRP                 0
dtype: int64
```

Imputing NULL with MEAN



```
sales_mean = df['SALES'].mean()
sales_mean
```

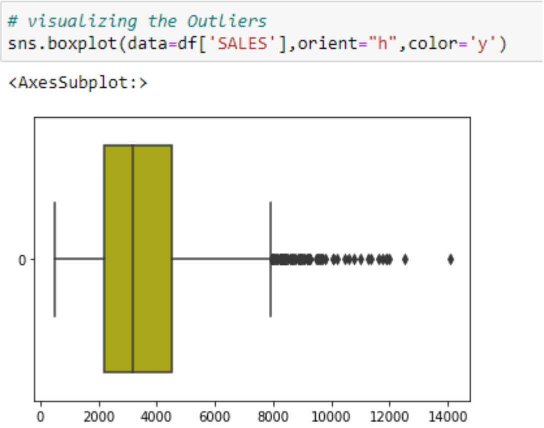
```
3551.433111032281
```

```
df['SALES'].fillna(value=sales_mean,inplace=True)
```

```
df['SALES'].isnull().sum()
```

```
0
```

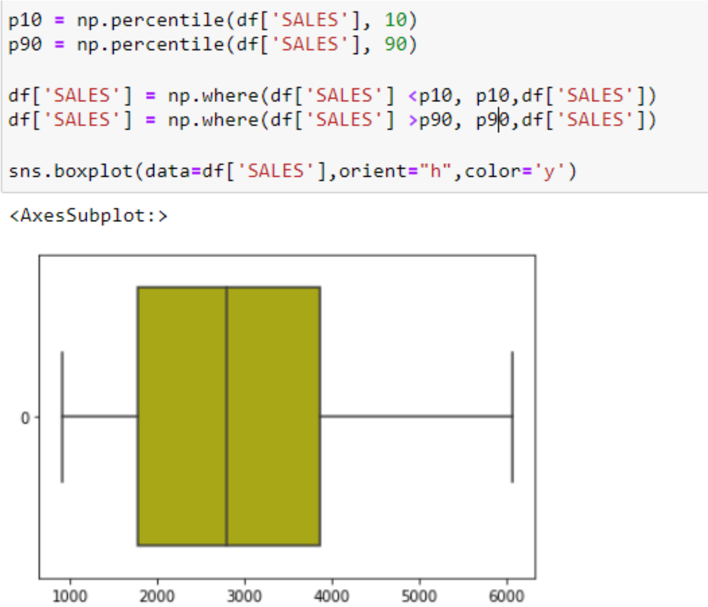
8. Outliers
   a. Check for outliers
   b. visualization of outliers
   c. Treat the outliers



```
# visualizing the Outliers
sns.boxplot(data=df['SALES'],orient="h",color='y')
```

```
<AxesSubplot:>
```

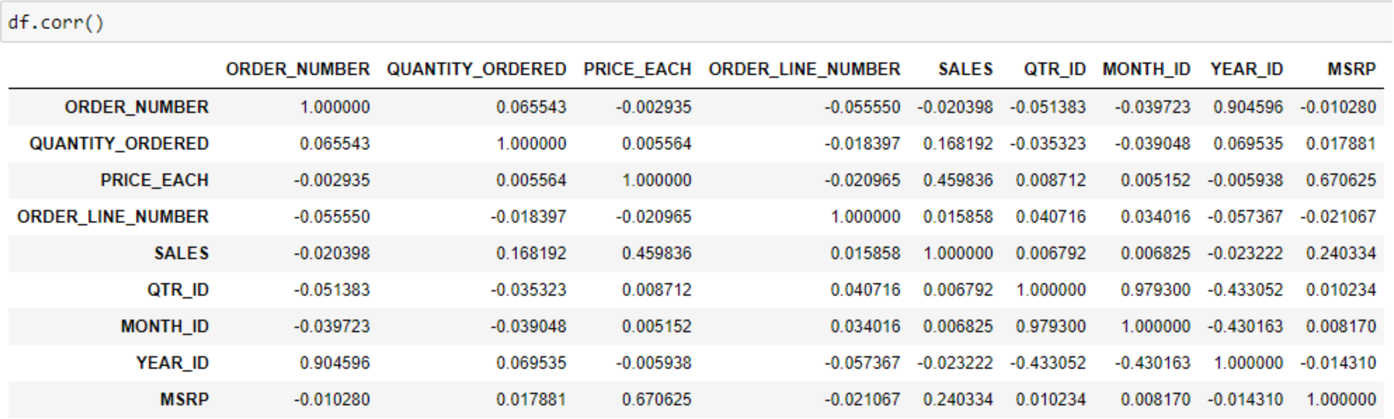
Quantile based flooring & capping



```
p10 = np.percentile(df['SALES'], 10)
p90 = np.percentile(df['SALES'], 90)

df['SALES'] = np.where(df['SALES'] <p10, p10,df['SALES'])
df['SALES'] = np.where(df['SALES'] >p90, p90,df['SALES'])

sns.boxplot(data=df['SALES'],orient="h",color='y')
```

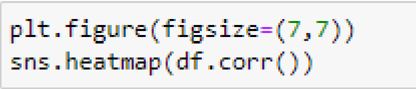
```
<AxesSubplot:>
```

9. Correlation
   a. Correlation matrix
   b. Heatmap for correlation

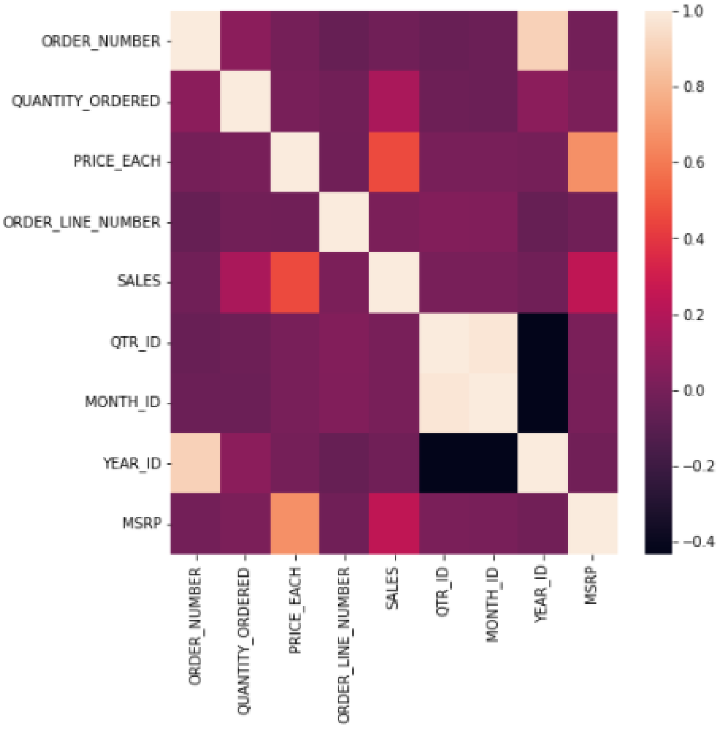


```
df.corr()
```

	ORDER_NUMBER	QUANTITY_ORDERED	PRICE_EACH	ORDER_LINE_NUMBER	SALES	QTR_ID	MONTH_ID	YEAR_ID	MSRP
ORDER_NUMBER	1.000000	0.065543	-0.002935	-0.055550	-0.020398	-0.051383	-0.039723	0.904596	-0.010280
QUANTITY_ORDERED	0.065543	1.000000	0.005564	-0.018397	0.168192	-0.035323	-0.039048	0.069535	0.017881
PRICE_EACH	-0.002935	0.005564	1.000000	-0.020965	0.459836	0.008712	0.005152	-0.005938	0.670625
ORDER_LINE_NUMBER	-0.055550	-0.018397	-0.020965	1.000000	0.015858	0.040716	0.034016	-0.057367	-0.021067
SALES	-0.020398	0.168192	0.459836	0.015858	1.000000	0.006792	0.006825	-0.023222	0.240334
QTR_ID	-0.051383	-0.035323	0.008712	0.040716	0.006792	1.000000	0.979300	-0.433052	0.010234
MONTH_ID	-0.039723	-0.039048	0.005152	0.034016	0.006825	0.979300	1.000000	-0.430163	0.008170
YEAR_ID	0.904596	0.069535	-0.005938	-0.057367	-0.023222	-0.433052	-0.430163	1.000000	-0.014310
MSRP	-0.010280	0.017881	0.670625	-0.021067	0.240334	0.010234	0.008170	-0.014310	1.000000



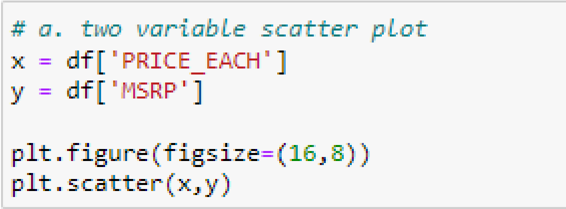
```
plt.figure(figsize=(7,7))
sns.heatmap(df.corr())
```



10. Graphical representation

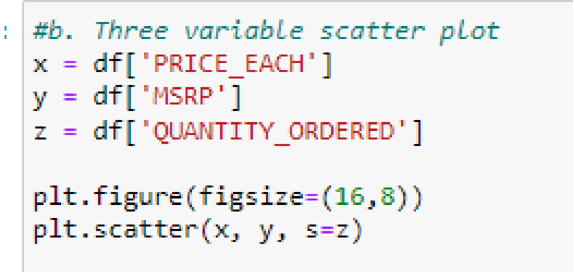
a. Scatter plot

i. Two Variable



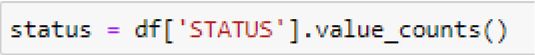
```
# a. two variable scatter plot
x = df['PRICE_EACH']
y = df['MSRP']

plt.figure(figsize=(16,8))
plt.scatter(x,y)
```

ii. Three Variable

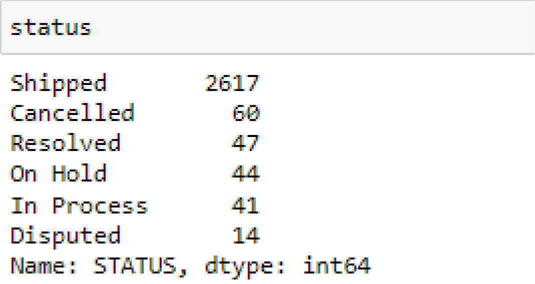


```
#b. Three variable scatter plot
x = df['PRICE_EACH']
y = df['MSRP']
z = df['QUANTITY_ORDERED']

plt.figure(figsize=(16,8))
plt.scatter(x, y, s=z)
```

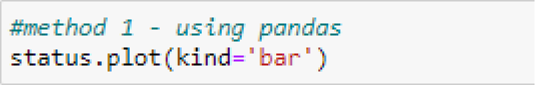
b. Bar graph



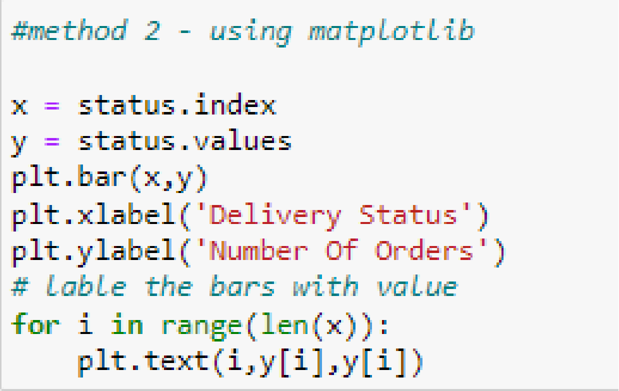
```
status = df['STATUS'].value_counts()
```



```
status

Shipped       2617
Cancelled       60
Resolved        47
On Hold         44
In Process      41
Disputed        14
Name: STATUS, dtype: int64
```

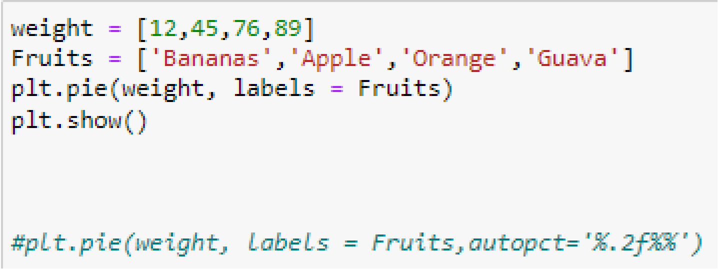


```
#method 1 - using pandas
status.plot(kind='bar')
```

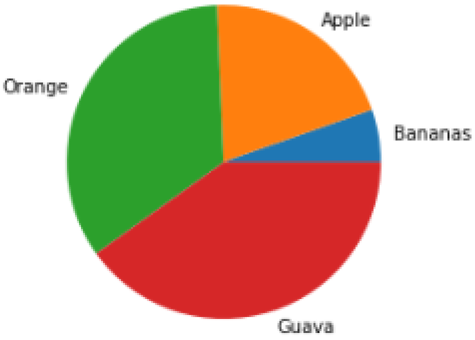


```
#method 2 - using matplotlib

x = status.index
y = status.values
plt.bar(x,y)
plt.xlabel('Delivery Status')
plt.ylabel('Number Of Orders')
# lable the bars with value
for i in range(len(x)):
    plt.text(i,y[i],y[i])
```

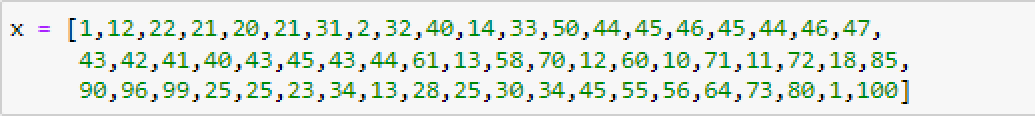
c. Pie chart



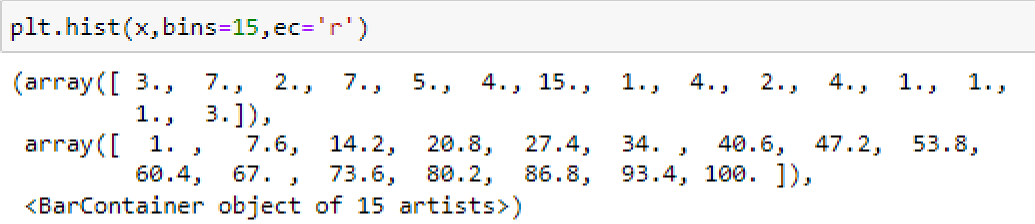
```
weight = [12,45,76,89]
Fruits = ['Bananas','Apple','Orange','Guava']
plt.pie(weight, labels = Fruits)
plt.show()



#plt.pie(weight, labels = Fruits,autopct='%.2f%%')
```



d. Histogram

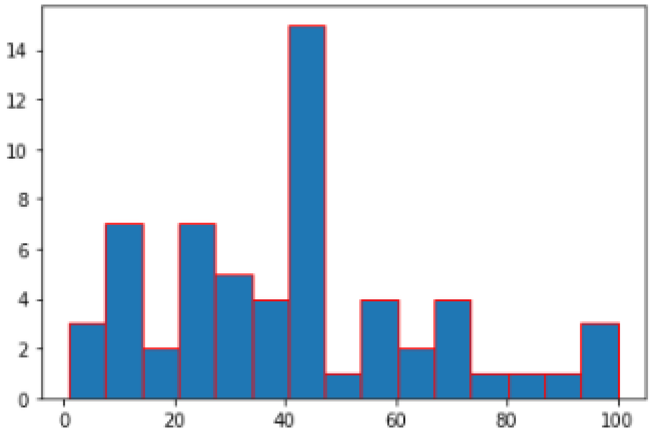


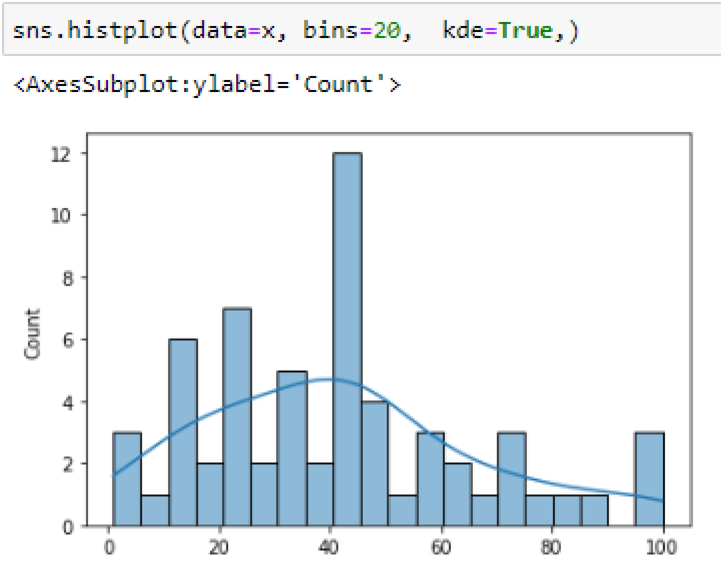
```
x = [1,12,22,21,20,21,31,2,32,40,14,33,50,44,45,46,45,44,46,47,
     43,42,41,40,43,45,43,44,61,13,58,70,12,60,10,71,11,72,18,85,
     90,96,99,25,25,23,34,13,28,25,30,34,45,55,56,64,73,80,1,100]
```



```
plt.hist(x,bins=15,ec='r')
```

```
(array([ 3.,  7.,  2.,  7.,  5.,  4., 15.,  1.,  4.,  2.,  4.,  1.,  1.,
         1.,  3.]),
 array([  1. ,   7.6,  14.2,  20.8,  27.4,  34. ,  40.6,  47.2,  53.8,
         60.4,  67. ,  73.6,  80.2,  86.8,  93.4, 100. ]),
 <BarContainer object of 15 artists>)
```





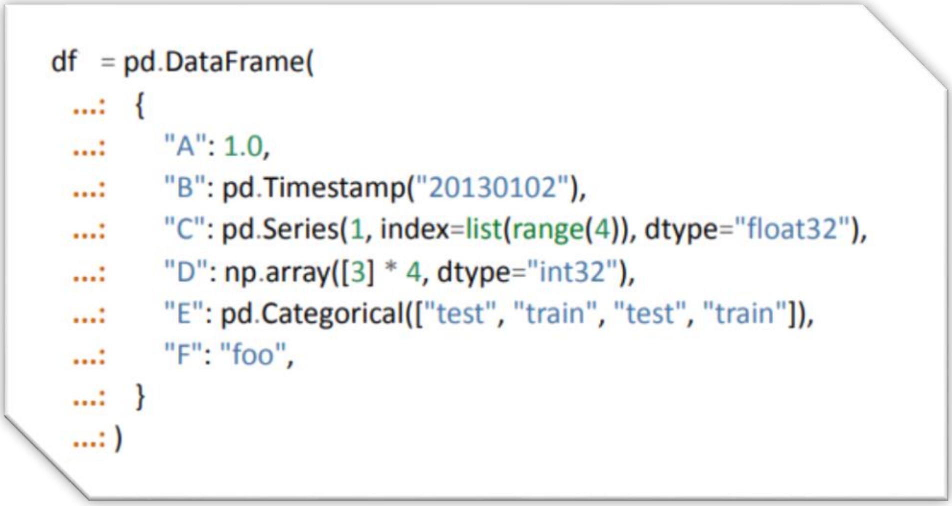
```
sns.histplot(data=x, bins=20,  kde=True,)
```

```
<AxesSubplot:ylabel='Count'>
```

Important Operation on dataset using Pandas

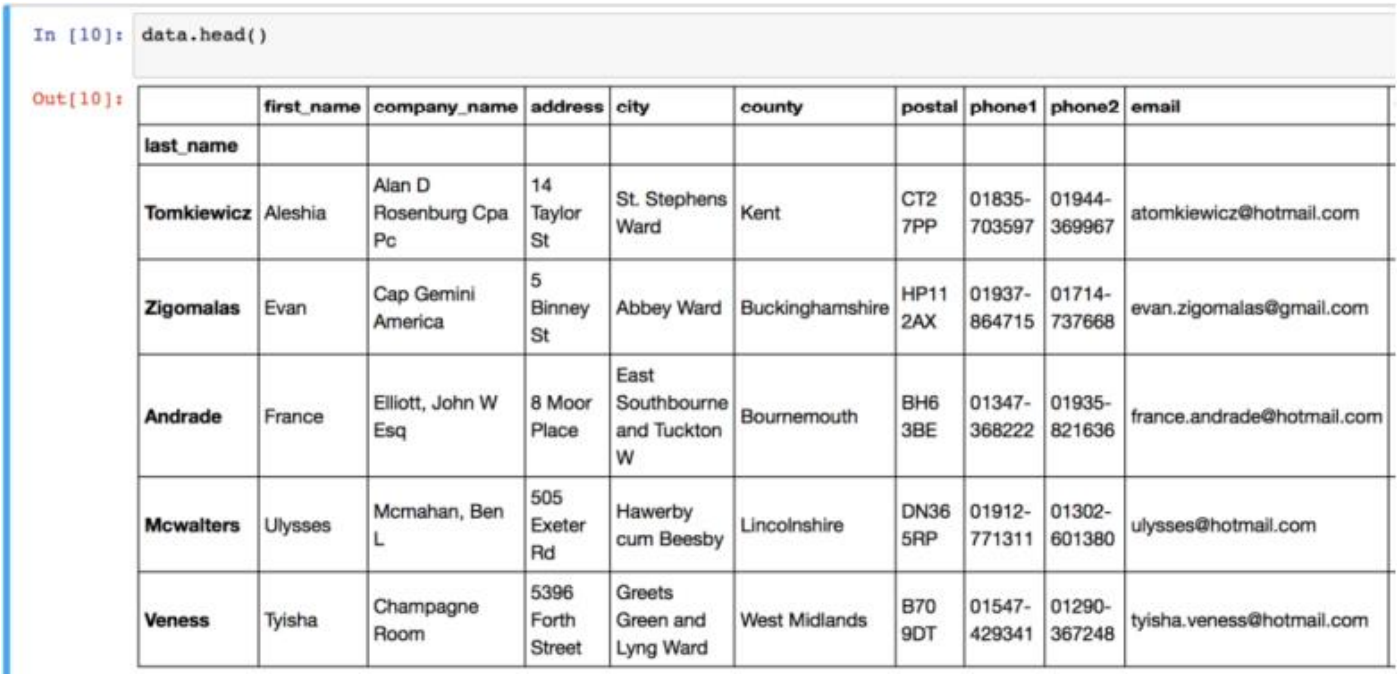
10-minute guide for pandas Link

1. Creation of dataset



```
df  = pd.DataFrame(
   ...:  {
   ...:      "A": 1.0,
   ...:      "B": pd.Timestamp("20130102"),
   ...:      "C": pd.Series(1, index=list(range(4)), dtype="float32"),
   ...:      "D": np.array([3] * 4, dtype="int32"),
   ...:      "E": pd.Categorical(["test", "train", "test", "train"]),
   ...:      "F": "foo",
   ...:  }
   ...: )
```

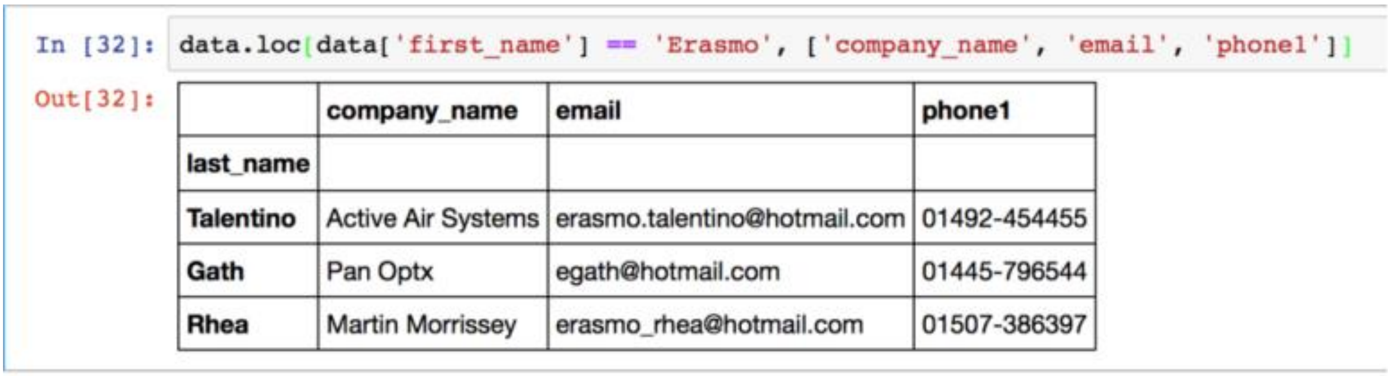
2. Filtering, selecting by Boolean mask and index



```
In [10]: data.head()
```

Out[10]:

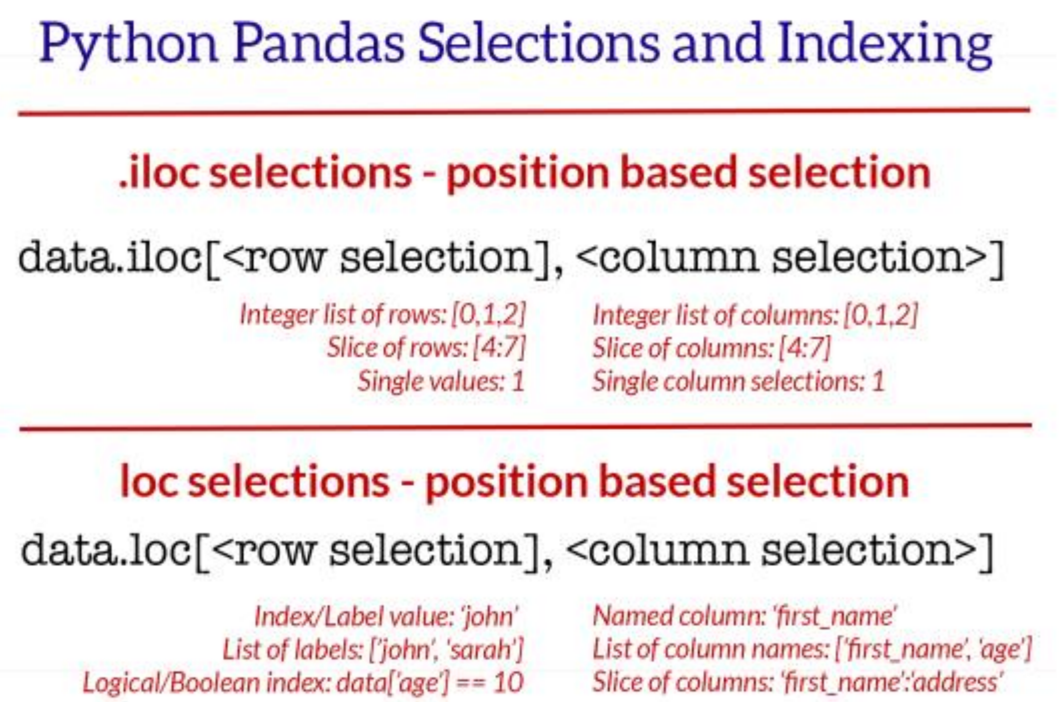
	first_name	company_name	address	city	county	postal	phone1	phone2	email
last_name									
Tomkiewicz	Aleshia	Alan D Rosenburg Cpa Pc	14 Taylor St	St. Stephens Ward	Kent	CT2 7PP	01835-703597	01944-369967	atomkiewicz@hotmail.com
Zigomalas	Evan	Cap Gemini America	5 Binney St	Abbey Ward	Buckinghamshire	HP11 2AX	01937-864715	01714-737668	evan.zigomalas@gmail.com
Andrade	France	Elliott, John W Esq	8 Moor Place	East Southbourne and Tuckton W	Bournemouth	BH6 3BE	01347-368222	01935-821636	france.andrade@hotmail.com
Mcwalters	Ulysses	Mcmahan, Ben L	505 Exeter Rd	Hawerby cum Beesby	Lincolnshire	DN36 5RP	01912-771311	01302-601380	ulysses@hotmail.com
Veness	Tyisha	Champagne Room	5396 Forth Street	Greets Green and Lyng Ward	West Midlands	B70 9DT	01547-429341	01290-367248	tyisha.veness@hotmail.com



```
In [32]: data.loc[data['first_name'] == 'Erasmo', ['company_name', 'email', 'phone1']]
```

Out[32]:

	company_name	email	phone1
last_name			
Talentino	Active Air Systems	erasmo.talentino@hotmail.com	01492-454455
Gath	Pan Optx	egath@hotmail.com	01445-796544
Rhea	Martin Morrissey	erasmo_rhea@hotmail.com	01507-386397



Python Pandas Selections and Indexing

.iloc selections - position based selection

data.iloc[<row selection>, <column selection>]

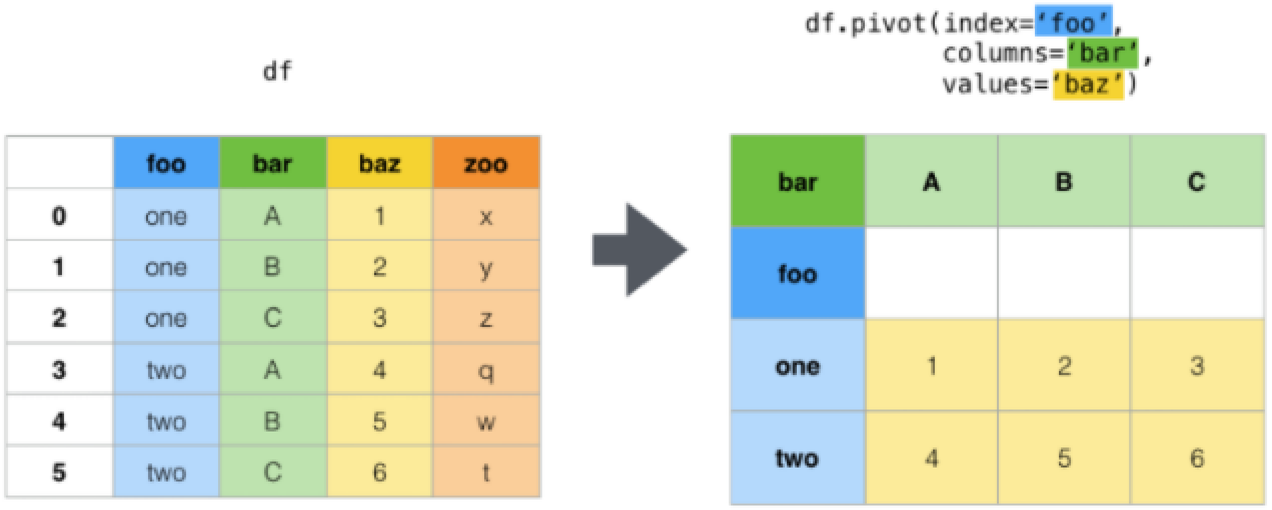
Integer list of rows: [0,1,2]	*Integer list of columns: [0,1,2]*
Slice of rows: [4:7]	*Slice of columns: [4:7]*
Single values: 1	*Single column selections: 1*

loc selections - position based selection

data.loc[<row selection>, <column selection>]

Index/Label value: 'john'	*Named column: 'first_name'*
List of labels: ['john', 'sarah']	*List of column names: ['first_name', 'age']*
Logical/Boolean index: data['age'] == 10	*Slice of columns: 'first_name':'address'*

3. Pivot and Melt

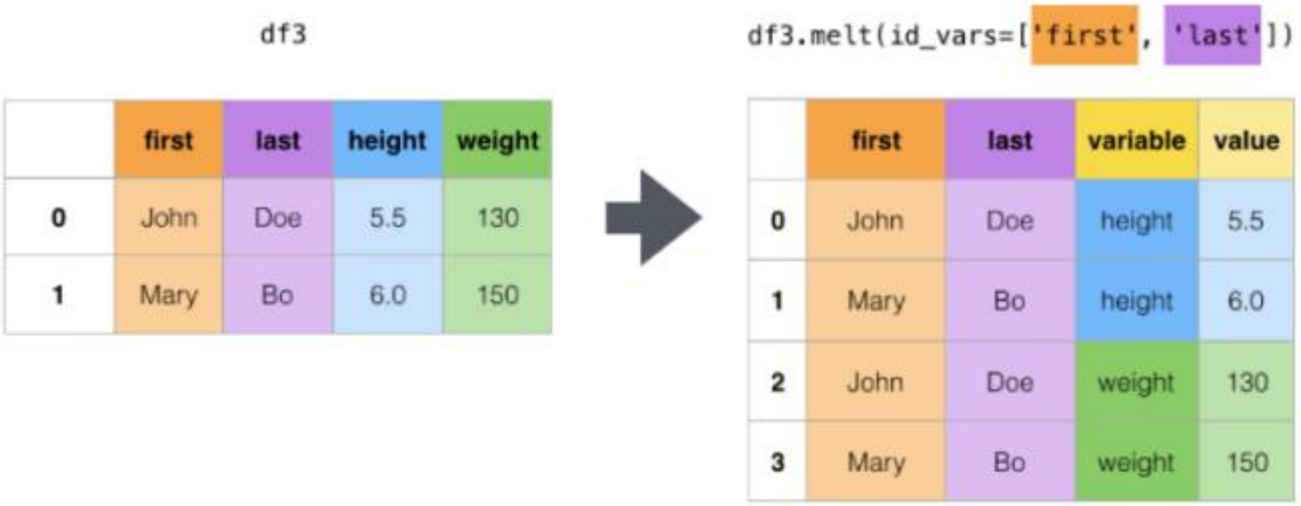


df

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

```
df.pivot(index='foo',
         columns='bar',
         values='baz')
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6



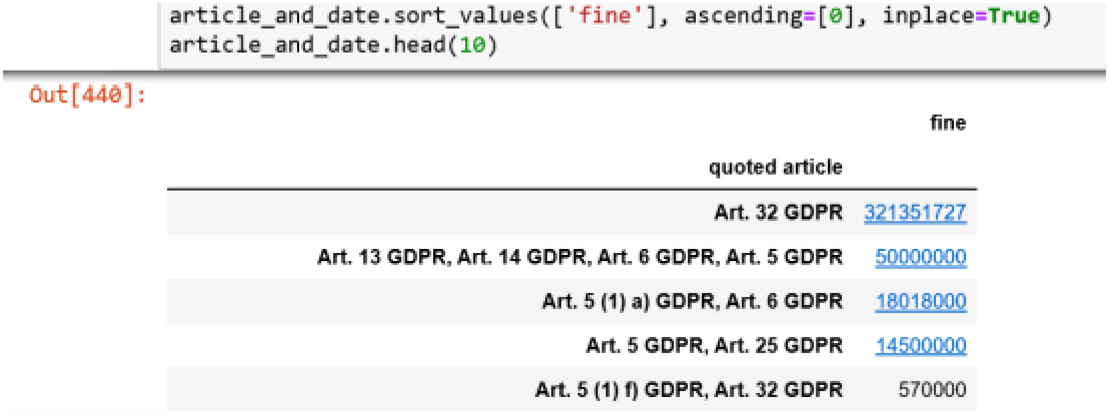
df3

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

```
df3.melt(id_vars=['first', 'last'])
```

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

4. Sorting

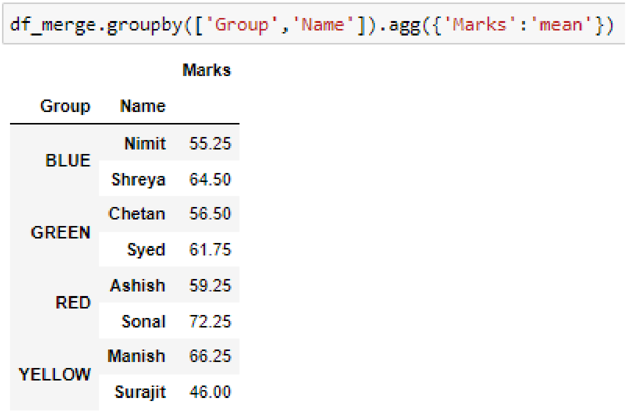


```
article_and_date.sort_values(['fine'], ascending=[0], inplace=True)
article_and_date.head(10)
```

Out[440]:

quoted article	fine
Art. 32 GDPR	321351727
Art. 13 GDPR, Art. 14 GDPR, Art. 6 GDPR, Art. 5 GDPR	50000000
Art. 5 (1) a) GDPR, Art. 6 GDPR	18018000
Art. 5 GDPR, Art. 25 GDPR	14500000
Art. 5 (1) f) GDPR, Art. 32 GDPR	570000

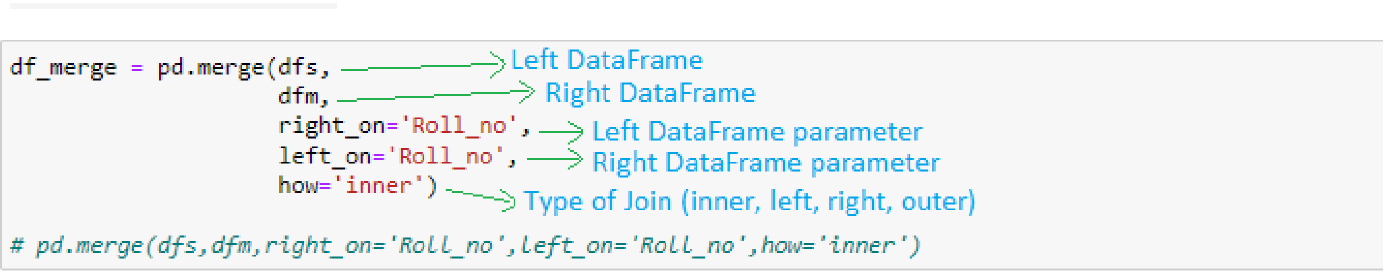
5. Group by



```
df_merge.groupby(['Group','Name']).agg({'Marks':'mean'})
```

		Marks
Group	Name	
BLUE	Nimit	55.25
	Shreya	64.50
GREEN	Chetan	56.50
	Syed	61.75
RED	Ashish	59.25
	Sonal	72.25
YELLOW	Manish	66.25
	Surajit	46.00

6. Joins



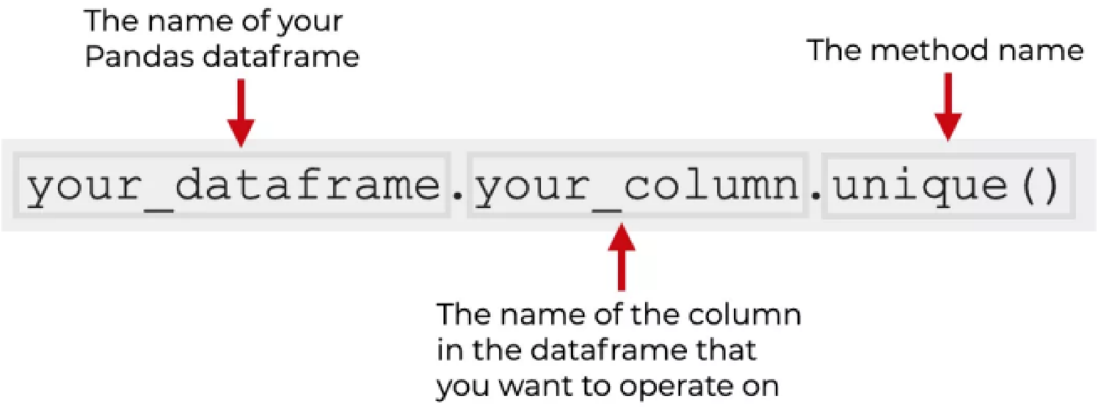
```
df_merge = pd.merge(dfs,
                    dfm,
                    right_on='Roll_no',
                    left_on='Roll_no',
                    how='inner')
# pd.merge(dfs,dfm,right_on='Roll_no',left_on='Roll_no',how='inner')
```

Left DataFrame
Right DataFrame
Left DataFrame parameter
Right DataFrame parameter
Type of Join (inner, left, right, outer)

7. Timeframe check - df.date.unique()

df.date.unique()

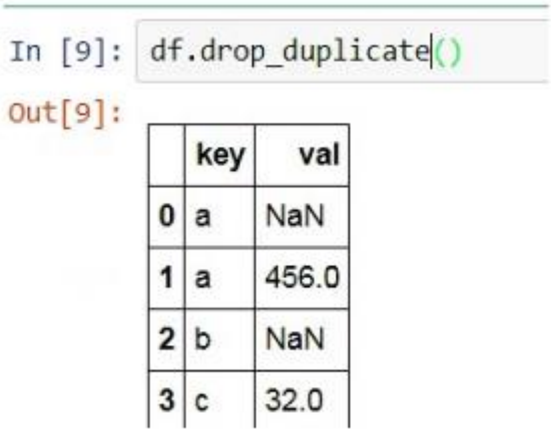
[1 jan, 2,3...30 Jan]



```
your_dataframe.your_column.unique()
```

The name of your Pandas dataframe
The method name
The name of the column in the dataframe that you want to operate on

8. Drop Duplicates

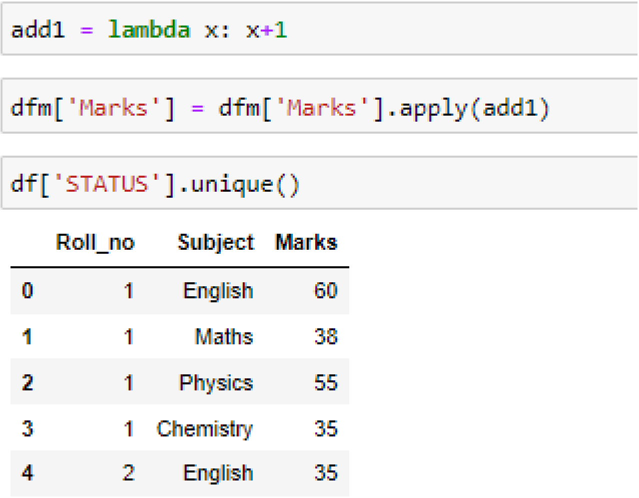


```
In [9]: df.drop_duplicate()
```

Out[9]:

	key	val
0	a	NaN
1	a	456.0
2	b	NaN
3	c	32.0

9. Pandas UDF - .apply



```
add1 = lambda x: x+1
```

```
dfm['Marks'] = dfm['Marks'].apply(add1)
```

```
df['STATUS'].unique()
```

	Roll_no	Subject	Marks
0	1	English	60
1	1	Maths	38
2	1	Physics	55
3	1	Chemistry	35
4	2	English	35