Data Analysis with Python

- inplace = True → changes DF
- axis = 0 → entire row
- axis = 1 → entire column

1) Import libraries

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

2) Import Datasets

1) `df = pd.read_csv(r"PathName"/"fileName")` → should be in SAME directory
   - header = None, names = " "
   - encoding = 'unicode_escape' → if encoding error occurs
2) `df.shape` → gives no. of rows & columns
3) `df.head()` → top values
4) `df.tail()` → bottom values

3) Data cleaning

Info for Data cleaning: 1) `df.info()`

- Null values T/F: 2) `df.isnull()` or
- Sum of all Null values: `df.isnull().sum().sum()`

Removes NaN (missing)

- Deletes columns with 0 values in df.info(): 3) `df.drop(['colName'], axis = 1, inplace = True)`
- Removes Null values: 4) `df.dropna()`
- Removes entire row: 5) `df2 = df.dropna(how = 'any')` · If any NaN values; `df2 = df.dropna(how = 'all')` · If all NaN values

Remove duplicates

- Sum-duplicate values: 6) `df['colName'] = df['colName'].duplicated().sum()`
- removes duplicates: 7) `df.drop_duplicated()`

fill NaN (missing)

- fill NaN: 8) `df2 = df.fillna(value = 0)`
- fill NaN of sp. col: 9) `df['colName'].fillna('unknown', inplace = True)`
- fill NaN with Prev. val: 10) `df2 = df.fillna(method = 'pad')` } Row wise
- fill NaN will forw. val: 11) `df2 = df.fillna(method = 'bfill')` }

  axis = 1 for column wise
- fill NaN in diff. col: 12) `df2 = df.fillna({'colName1': 'value1', 'colName2': 'value2'})`
- Fill NaN with Mean: 13) `df2 = df.fillna(value = df['colName'].mean)`

Replace values

- replaces multiple values: 14) `df.loc[rowNo, 'colName'] = 'New value'`
- removes multiple values: 15)
```
for x in df.index:
    If df.loc[x, 'colName'] = 'old value':
        df.loc[x, 'colName'] = 'New value'   or
        df.drop[x, inplace = True]
```
- replace any/NaN value: 16) `df2 = df.replace(to_replace = np.nan, value = '0')` or any value
- Converts value from numeric to string: 17)
```
def conv(values):
    if values == 0:
        return 'no'
    else:
        return 'yes'
df['colName'] - df['colName'].apply(conv)
```
- Renames col Name: 18) `df.rename(columns = {'oldValue': 'NewValue'})`
- Describe col: 19) `df.describe()` or 20) `df[['colName']].describe()`

Change Data Type

- Find the datatypes — 21) `df.dtypes`
- changes Datatype(date) — 22) `df['Date'] = pd.to_datetime(df['Date'])`
- changes Date format to YYYY-mm-DD — 23) `df['Date'] = pd.to_datetime(df['Date'].strftime('%Y-%m-%d')`
- Change 11(other) — 24) `df['ColName'] = df['ColName'].astype('float')`

Save file

- Save cleaned file — 25)

```
import os
directory_path = 'C:/Users/shagu/Documents/Data Analyst/SQL'

csv_path = os.path.join(directory_path, 'practice1.csv')
df.to_csv(csv_path, index=False)
print(f"Table data saved to {csv_path}")
```

```
Table data saved to C:/Users/shagu/Documents/Data Analyst/SQL\practice1.csv
```

- To Know current CSV Path — 26) `os.getcwd()`
- Find list of ColNames — 27) `df.columns.tolist()`
- Merge 2 tables — 28) `pd.merge(df1, df2, on='ColName', how='inner')` (ColName: custID)

Missing Values

1) find % of NaN values

`df.isnull().mean() * 100`

2) If below 5%

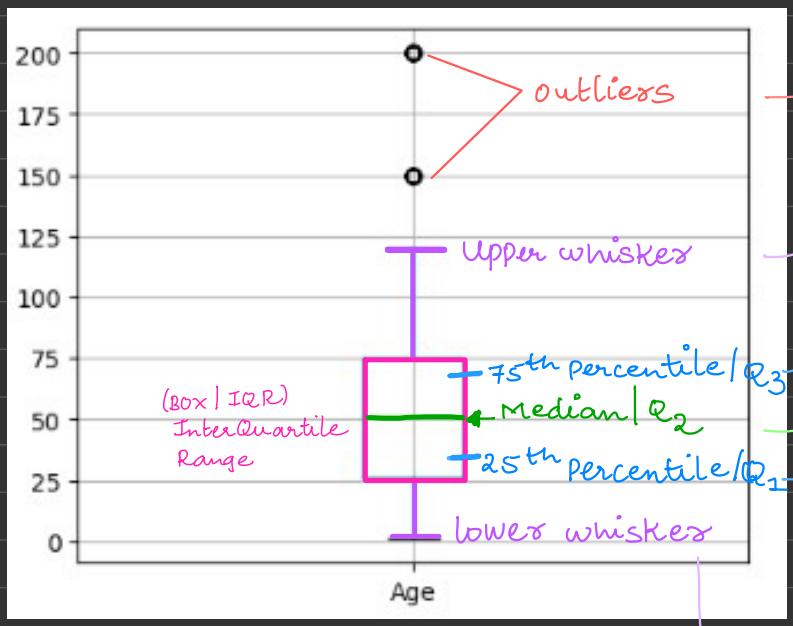
MCAR

- i) CCA
- ii) SimpleImputation (univariate)
  - Numerical
    - i) Mean
    - ii) Median
    - iii) Arbitrary
  - categorical
    - i) Mode
    - ii) Arbitrary

If above 5%

- MCAR / MAR → KNN
- MNAR → Iterative

Box plot → detects outliers



→ Data points that fall outside 1.5 times the IQR — Age = 150 & 200

→ Starts at Q_3 & extends upto largest non-outliers

→ the value below which 75% data lies — Age = 75

→ Middle value of dataset — Age = 50

→ the value below which 25% data lies — Age = 25

→ Starts at Q_1 & extends upto smallest non-outlier

IQR – a rectangular box → represents middle 50% of data

Whiskers → Vertical lines extending from the box & Not considered outliers.

Outliers → plotted individually as dots or circles

4) EDA (Exploratory Data Analysis)

- Group By
- Data Binning
- Pearson correlation

```
gb = df.groupby(['colName1','colName2'])['colName3'].aggregate('count').reset_index(name='NewColName')
```
• Counts Non-Null rows only

```
gb = df.groupby(['colName1','colName2']).size().reset_index(name='NewColName')
```
• counts all rows

① Group by

Categorical values

```
gb = df.groupby('colName', as_index=False)['colName2'].sum().sort_values(by='colName2', ascending=False)
```

Numerical values

☆ `.sort_values(by='count', ascending=False)` → For sorting in descending

② Data Binning

```
bins = np.linspace(min(df['colName']), max(df['colName']), 4)
labels = ("low", "moderate", "high")
df['New_Col'] = pd.cut(df['colName'], bins=bins, labels=labels, include_lowest=True)
```

- 4 → creates equal space bins
- ↳ 1 more than no. of labels
- 1 no. fewer than bin

③ Pearson Coefficient

Converts DF to Float

1) df.corr()
2) df[['ColName', 'ColName2']].corr()
3) df.apply(pd.to_numeric, errors='coerce')
4) sns.heatmap(df)

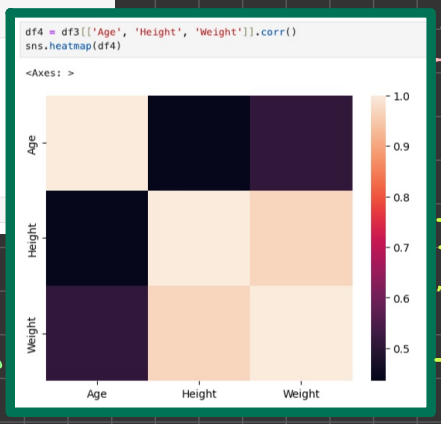
```
# Convert entire DataFrame to float, non-convertible values become NaN

df4 = df3.apply(pd.to_numeric, errors='coerce').corr()
df4
```

	name	Age	Height	Weight
name	NaN	NaN	NaN	NaN
Age	NaN	1.000000	0.435143	0.509119
Height	NaN	0.435143	1.000000	0.967105
Weight	NaN	0.509119	0.967105	1.000000

```
sns.heatmap(df4)
```

```
df4 = df3[['Age', 'Height', 'Weight']].corr()
sns.heatmap(df4)
```



more +ve value & light colour → more +ve relationship

blw 312 – 1810.33

blw 1810.33 – 3308.66

blw 3308.66 – 4807

```
bins = np.linspace(min(state['Orders']), max(state['Orders']),4)
labels = ("Min", "Moderate", "Max") → 4-1 = 3 labels
state['Status'] = pd.cut(state['Orders'], bins = bins, labels = labels, include_lowest = True)
state
```

4 equal space

312, 1810.33, 3308.66, 4807
min max

	State	Orders	Status
14	Uttar Pradesh	4807	Max
10	Maharashtra	3810	Max
7	Karnataka	3240	Moderate
2	Delhi	2740	Moderate
9	Madhya Pradesh	2252	Moderate
0	Andhra Pradesh	2051	Moderate
5	Himachal Pradesh	1568	Min
8	Kerala	1137	Min
4	Haryana	1109	Min
3	Gujarat	1066	Min
1	Bihar	1062	Min
6	Jharkhand	953	Min
15	Uttarakhand	824	Min
12	Rajasthan	555	Min
11	Punjab	495	Min
13	Telangana	312	Min

If NOT: then manually enter bin edges (1 more than labels)

- close to +1 = +ve relationship x↑, y↑
- close to −1 = −ve relationship x↑, y↓
- close to 0 = No relation

Libraries

Seaborn
1) sns.load_dataset(" ")
2) sns.countplot(data = df, x = ' ')
3) sns.barplot(data = df, x = ' ', y = ' ')
4) sns.lineplot( )
5) sns.displot( )
6) sns.scatterplot( )
7) sns.histplot( )

Matplotlib
1) plt.plot(x = ' ', y = ' ')
2) plt.title(' ')
3) plt.xlabel(' ')
4) plt.ylabel(' ')
5) plt.grid( )
6) plt.grid(axis = 'x') · only on x-axis
7) plt.show( )
8) plt.legend(title = ' ') or plt.legend( )

line 9) plt.plot( )
bar 10) plt.bar( )
Horiz. bar 11) plt.barh( )
pie 12) plt.pie( )
Scatter 13) plt.scatter(c = colors, cmap = ' ') viridis, green etc.
histogram 14) plt.hist( )
15) plt.subplot(nrows, ncolumns, index)
Saves chart 16) plt.savefig("chart.png")
clears figure 17) plt.cls( )
density plot 18) plot(kind = 'kde')

Parameters
- hue = Category
- palette = colour
- bins = binning
- KDE = For density

Labels the bars

```
ax = sns.barplot( )
For bars in ax.containers:
    ax.bar_label(bars)
```

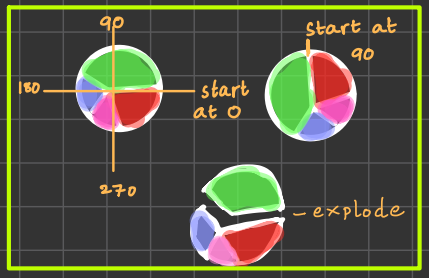
figure size
1) sns.set({'figure.figsize': (15, 5)})
2) plt.figure(figsize = 5, 5)

markers = '*'
linestyle = '--'
linewidth = 2
colors = 'r'

```
plt.legend(title = ' ',
    bbox_to_anchor = (1.05, 1),
    loc = 'upper left')
```

plt.subplot(2, 1, 1)
2 rows, 1st column, 1st index

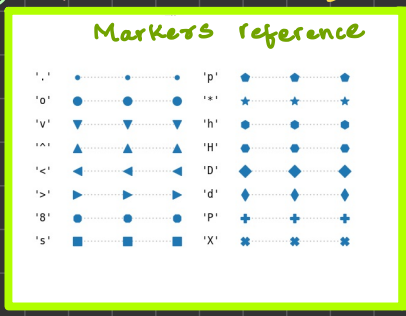
```
x = ['ColName']
mylabels = [' ', ' ', ' ',]
mycolors = ['r', 'hotpink', 'b', '4CAF50']
myexplode = [0.2, 0, 0, 0]
```



**

Code

```
plt.figure(figsize=(4,4))
sns.pie(plt.pie(x=' ', labels=mylabels, colors=mycolors, autopct='%1.2F%%', startangle=90, explode=myexplode)
plt.title(' ')
plt.legend(title=' ')
plt.xlabel(' ')
plt.ylabel(' ')
plt.xticks(rotation=(15))
plt.show()
```




```
# Colour reference
'r'-red  'b'-blue  'g'-green
'm'-magenta  'c'-cyan
'y'-yellow  'k'-black
'w'-white

# Line reference
'-'  '--'  '-.'  ':'
```

Pandas

- directory to be same
- 1st 4 rows
- 1st and 5th row
- Column of 2nd row
- Column as DataFrame
- set no. of rows

```
1) df = pd.read_csv(r"Path name"/"filename")
2) df.iloc[[0:4]]
3) df.iloc[[0, 4]]
4) df.loc[RowNo, 'ColName']
5) df[['ColName']]
6) pd.options.display.max_rows = 10,000
7) df.head()
8) df.info()
9) df.describe()
10) df.rename(columns={'oldValue':'NewValue'})
11) df.corr()
```

- Converts JSON file to DataFrame

```
12) df = pd.read_json("file path"/"file name")
    df.to_string()
```

- Saves as CSV file
- Converts to DataFrame

```
13) df.to_csv()
14) Pd.DataFrame()
```

Numpy

- Changes do Not affect
- Changes affect
- reshapes into rows-col
- rearrange to array
- Creates an array
- A range of no. as array
- evenly spaced no.
- multiplicath of matrix

```
1) arr.copy()   } Owns the
2) arr.view()   } data
3) arr.reshape()  } mirror
4) arr.flattens() } image
5) arr = np.array()
6) np.arange()
7) np.linspace()
8) np.dot()
9) np.transpose()
```

- concats the string
- replace the value

```
10) np.char.add()
11) np.char.replace()
12) np.char.upper()
13) np.char.lower()
```

- splits where space
- splits where newline

```
14) np.char.split()
15) np.char.splitlines()
```

```
16) np.mean()
17) np.median()
18) np.std()
19) np.sum()
20) np.min()/max()
```

Speed = [98, 97, 95, 96, 103]

mean = $\frac{98+97+95+96+103}{5}$

median = 95, 96, (97), 98, 103

(sort the ascending)

odd → central value

even → mean of 2 central values

subplot (overlapped)

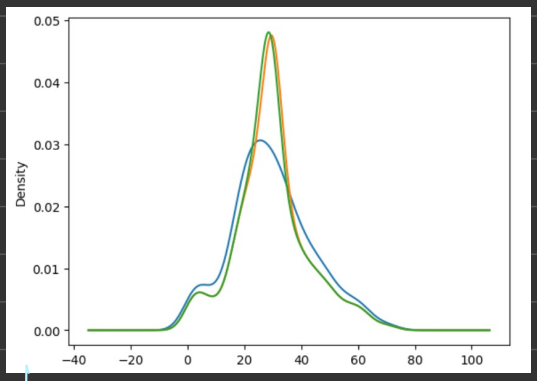
different than this
subplot(1, 2, 1) → Index
nrows ncols
= = 11

① fig = plt.figure()
ax = fig.add_subplot(111)

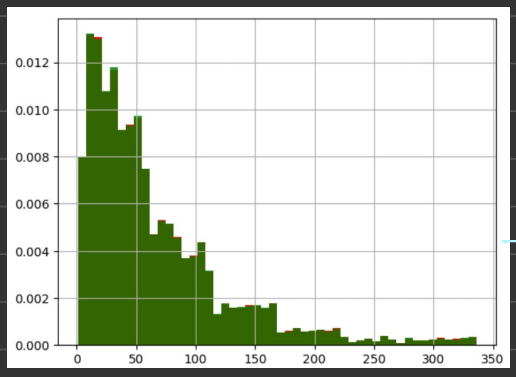
df['Age'].plot(kind='kde', ax=ax)
df['Age_mean'].plot(kind='kde' ax=ax)
df['Age_median'].plot(kind='kde ax=ax)
} can select colour as well

② fig = plt.figure()
ax = fig.add_subplot(111)

df['colName'].hist(bins=50, ax=ax, density=True, color='red')
new_df['colName'].hist(bins=50, ax=ax, density=True, color='red')



variation



overlaps (No diff)

subplots (grid of plots)

```
cols = ['colName1', 'colName2']
fig, axes = plt.subplots(2, 2, figsize=(12,8))
                         nrows ncols
axes = axes.flatten()
for i, cols in enumerate(cols):
    df[col].hist(bins=50, ax=axes[i], density=True, alpha=0.2, color='red',
                 label='original-{col}')
    new_df[col].hist(bins=50, ax=axes[i], density=True, alpha=0.2, color='red',
                     label='imputed-{col}')
    axes[i].x_label(col)
    axes[i].y_label()
    axes[i].legend()

    plt.tight_layout()
    plt.show()
```