Pandas

Introduction to Pandas

▼ **Lambdas**

Earlier:

`df[col].str.upper()` [[note: we can use df.columns or df.columns.values]]

Now:

`df[col].apply(lambda x:x.upper())`

Called anonymous functions

Used mostly on dataframes as one liners

```
# Syntax:
# lambda arguments:return statement
eg,fn=lambda x:y:x+y
fn(2,4)
```

Use case:

as argument of sort()

```
ls=['sahil','sonia-choudhary','shubham','deepak']
# sort this list based on length of values
# custom.Not allready available in python =>make fn=>anon. fn
# sort function accepts key argument to make custom sorting
#and this key accepts a function
ls.sort(key=lambda x:len(x))
```

Sales	Price	Perc
$24.89	$23789	25.8%
$45,555,55.00	$45999	33.8%

For %:

`df['Perc'].str.rstrip('%').astype(float)/100`

For comma, $:

```
cols=[Sales,Price]
for column in cols:
  # remove $
  df[column]=df[column].apply(lambda x : x[1:] if x.startswith('$') else x)

  # remove ,
  df[column]=df[column].apply(lambda x : x.replace(',',' ') if ',' in x else x) # apply means iterate through each element

Now convert to numeric→
for c in cols:
  df[c]=pd.to_numeric(df[c])
```

▼ **Playing with Dates and Times**

Python has Datetime lib to **create** dates/times

import datetime

```
# summary
datetime.date() # pass year,month,day
datetime.time() #pass h,m,s,ms
datetime.datetime() # pass,month,day,hour,minute,second,ms
datetime.timedelta()
```

▼ Date

Create a date:

Create any date:

`d =datetime.date(2021,7,24)`

Create today's date:

`today=datetime.date.today()`

💡 Note: If you want to specify timezone,use .now() instead of today()

💡 Tip: Don't add leading leading zeroes yourself, it will cause syntax error

When date is created, you will need to get its components too so next section is to get these

Get Date's components:

`Get year -d.year`

`Get month - d.month`

`Get day - d.day`

`Get weekday - d.isoweek()`

`eg,Monday-1,Tuesday-2

▼ Time

Create a Time:

`t=datetime.time(hour,min,sec,millisec)`

Get time:

`t.hour()`

`t.minute`

`t.seconds()`

▼ DateTime

Create a Date Time:

`dt=datetime.datetime(2016,7,12,2,30,45,1400)`

Get date: dt.date()

Get time: dt.time()

▼ TimeDeltas

Add to dates to get future dates

`7day=datetime.timedelta(days=7) # 7 dats after date`

Now do date(date)+7day(timedelta)

Note:

date+date = timeedelta

date+timedelta=date

▼ DateTime with Pandas

DF example

ID	Time column
01	2020/9/02 9:30
02	2020/9/02 9:30
03	2020/9/02 9:30

These '/' are not python format (unless converted using format argument)

So, first step should be to check if column is string or datetime

If not datetime, do this :

`df['Time column']=pd.to_datetime(df['Time column'])`

Now you will get the formatting as '-'

This - is python's format for dates

💡 Note: To specify date formats use format property

```
pd.to_datetime(df['Time column'],format=%d/%m/%Y)
pd.to_datetime(df['Time column'],format=%d-%m-%Y)
pd.to_datetime(df['Time column'],format=%d--%m--%Y)
```

Note:

% is used to specify each format code

%m → month

%d → day

%Y → year(4 digits)

If the column type is datetime, you can do these things with that column(just make sure to write .dt before the metric you want)

Name	Tags
df['Time column'].dt.hour	pulls hour for us
df['Time column'].dt.dayofyear	
df['Time column'].dt.year	
df['Time column'].dt.month	
df['Time column'].dt.day	
df['Time column'].dt.day	
df['Time column'].dt.week	
df['Time column'].dt.weekday	
df['Time column'].dt.weekday_name	Sunday/Monday
Untitled	
df['Time column'].dt.time	
df['Time column'].dt.hour	

Name	Tags
df['Time column'].dt.minute	
df['Time column'].dt.second	
Untitled	

Converting string to datetime

`Note: pd.todateime('1/1/2021') ⇒ 01-01-2021 00:00:00`

▼ Row Manipulation

```
df.loc[df['Time']≥ts,:]
# only show rows in which time>our custom time
```

▼ Range

Create a sequence of numbers

Range fn returns range object (which is iterable) ⇒convert it to list to see the inside elements

```
#Syntax
list(range(1,10))
# Here 1 is inclusive and 2 is exclusive
#By default step is 1

#To change the step
range(1,10,step)
```

Used in Loops to iterate 'n' no. of times

```
for i in range(1,3):
    print(i)    # 1,2
```

▼ Data Cleaning | EDA 101

Source: yt/J chares Tech

Column Cleaning:

df.head() → rows

df.columns → columns

Check what methods you can use on df col? → `dir(df.columns)`

Get columns as list:

`df.columns.tolist()`

Get summary of columns:

`df.columns.summary()`

Convert column names to series | df:

`df.columns.to_series()`

`df.columns.to_frame()`

Check if specific column is there or not:

`df.columns.contains('Name')` → True | False

Check if any duplicate column is there:

`df.columns.duplicated()` → False False False

Check methods/attributes of String:

`dir(df.columns.str)`

Make column names to lower case:

`df.columns.str.lower()`

Make column names to Upper case:

`df.columns.str.upper()` → Everything Big

Make column names to Title case:

`df.columns.str.title` → Camel Case

Make column names to Capitalize:

`df.column.str.capitalize()` → only first letter big

Replace empty spaces with underscores:

`df.columns.str.replace(' ','-')`

Rename columns:

`df.rename(columns={'oldname':'newname'},inplace=True)`

Check total number of columns:

`len(df.columns)`

Select particular columns:

`df.columns.values[0:4]`

Get 2nd column and rename it:

`df.columns.values[2]='DOB'`

Select all columns except one:

`df.colummns[df.column!  = 'colname']`

or

`df.loc[:,df.columns!  = 'name'].columns`

or

`df.loc[:,-df.columns.isin(['name1','name2']).columns`

Select column names that begins with particular word:

`a=df.columns.str.startswith('Street')` → gives bool array [true false false true]

`df.loc[:,a].` `columns` // to get only col names not values

Select group of column names:

`a=df.columns.` `values` `[[  0,3,5]]` # gives col1,col2,col3 # to get the Values also

or

`df.columns [ 0:5]` # gives col1,col2,col3

then, `df.loc[:,a]`

▼ **Playing with Missing Values**

Step 1: Detect the missing values

`missing_values=["N/A","na",np.nan(which is NaN)]`

By default, pandas only consider NaN as missing

It ignores na and N/a⇒ Specify custom Nan's while loading file

⇒`pd.read_csv('',` `na_values` `=missing_values)`

> 💡 Note: This na_values property is very useful

Step 2:Find the number of missing values in each column

`df.isnull.sum()` ⇒will give colname →missing values total in that column

`sns.heatmap(df.isnull(),yticklabels=False,annot=True)`⇒will plot the missing values

Step 3:Remove the values

`df.dropna() ⇒drop the entire row even if row has only 1 na item`

Note: It is similar to `df.dropna(axis='index',how='any')`

`df.dropna(how="all") ⇒drop the entire row only if all items of row are NaN`

Step 3:Fill those values with something

df.fillna(0) ⇒ Nan⇒0.0

df.fillna(method='ffill') ⇒ above cell's value will be copied here

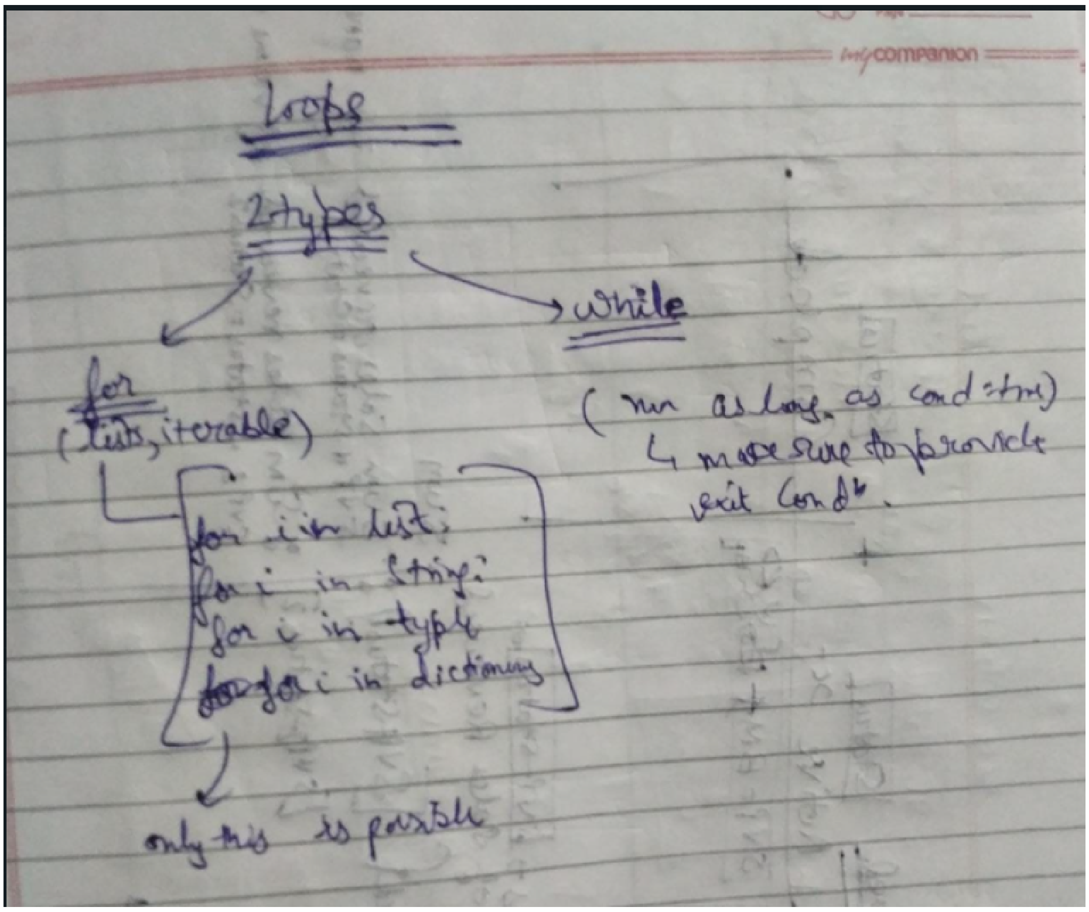
df.fillna(method='bfill') ⇒ below cell's value will be copied here

Interpolation:

```
df.interpolate() #takes the average of above and bottom value of nan's(above+below/2)
df.dropna(how='any',subset=['email'])  # Only look for email
df.replace('',np.nan,inplace=True)
```

▼ **Important Regex**

▼ **Loops**



▼ **Converting Data Types**

Source: yt/chartexplorer/change column data type

💡 Summary: Use pd.numeric only if mixed col(mixed with strings or mixed with Nan's)
Otherwise use astype

Use case 1: Numbers as strings

['1','2','3','4']

String_col

1

2

3

4

dtype=object

Note: String gets converted into object

`df['string_col']=df['string_col'].astype(int//float//string)`

`df['string_col']=pd.to_numeric(df['string_col']`

Use case 2: Ints+Floats

[1,2,3,4.6]

int_float

1.0

2.0

3.0

4.6

dtype=float

Note: if even 1 float ⇒all gets converted into float

Float⇒Int

`df['int_float'].astype(int)`

It rounds all values to down

ie, 4.1→4 , 4.6→4

Fix: first round them to nearest whole number

`df['int_float'].round(0).astype(int)`

Note: round 0 turns 4.1→4, 4.2→4 , 4.6→5

or

pd.to_numeric(df['int_float']) → gives float ⇒ Downcast to int(but downcast only work if .0)

so to get integers:

1.convert all float values to .0

2.Downcast to Integer

ie, `pd.to_numeric(df['int_float'].round(0),downcast='Integer')`

Use case 3: Strings + Ints

mix_col

sahil

-2

3

4

dtype=object

Note: String + nos gets converted to object `df['mix-col']=pd.to_numeric(df[..],errors='coerce')`

This errors=coerce converts strings to Nans and are ignored

But Nan's are floats⇒ Our entre col is flat now

⇒.astype('Int64') # It changes numpy's Nan's to pandas <Na> which are ints

Use case 4: Ints + Nans

[1,2,3,NaN]

missing

1.0

2.0

3.0

NaN

dtype=float

Note: even if 1 Nan⇒ nan is considered as float in pandas⇒ entire col gets converted to float

Use case 5: Strings(currencies)

money

$15,000.00

$12,500.98

`df['money'].replace('$','').replace(',','').to_numeric(..)`

Note: Replace can be chained

▼ **Data Type comparisons**

Python data types	Pandas data types
str/mixed	object
int	int
float	float
bool	bool
Na	datetime

Check Datatype:

type() → for 1

or

df.dtypes() → for all

Convert Datatype:

.astype(int|float|str)

.pd.to_numeric()

.pd.to_datetime()

▼ **Use Cases**

Notes: Entire columns can be converted to str and then we can do anything using str operations

id	name	variable

id	name	variable
1	A	Cases_ind
2	B	Cases_pak
3	C	Cases_uk

```
a=df['variable'].str.split('_')
# 0 [cases,ind]
# 1 [cases,pak]
# 2 [cases,uk]
# ⇒str[0]=⇒you now hve cases
# ⇒str[1]=⇒you now hv countrues
# so,assign theseto cls
df['new col 1']=a.str[0]
df['new col 2']=a.str[1]
# or
df[[ 'newcol1','newcol2']] =df['variable'].str('_',expand=True)
```

USE CASE 2:

name	job	salary
sahil#	dta$scientist	6500
sonia>	Student>	87$
%saurabh<<	student#%&	45%

Applying to 1 col:

```
df['name'].str.replace(r'\W','')
# note:r means start regex
# w means letters and numbers
# W means all except letters and numbers
# '' means string
# \ means escape these characters..there are not normal chars..they have a defined meaning
```

Applying to entire df:

```
for col in df.columns:
  df[col]=df[col].str.replace(r'\W','')
```

USE CASE 3:

full name	PHONE
Sahil choudhary	0112
Sonia choudhary	0212
Shubham choudhary	0312

```
first_namel=[]
last_namel=[]
col_name=df['fullname']

for name in col_name:
  fname,lname=name.split('',1)
  first_namel.append(fname)
```

```
    last_namel.append(lname)

df.insert(0,'First name',first nmel)
df.insert(0,'Last name',last nmel)
del df['fullname']
```

▼ Errors and Exceptions

Python error list

Name	Tags
Name error	If non existent property used
Value error	If wrong value is passed in Parameters, eg sqrt(-5)
Type error	If wrong type of parameter is sent to function
Key error	If non existing key is requested from dictionary
Attribute error	If property doesn't exist, eg obj.foo if foo doesn't exist

💡 Default arguments `fn(a,b-2)` are executed only when function is declared