PANDAS CHEAT SHEET

Installing, loading and checking version of pandas

```
!pip install pandas
print(pd.__version__)
```

1 dimentional data is called as Series

```
arr1 = np.array(range(1,11,2))
s = pd.Series(arr1)
s
```

2 dimentional data is called as DataFrame

EX1:

```
dic1 = {
    'name' : ['pramodha','keerthika','chandana','swathi','deepthi','siri','thaheera'],
    'place' : ['ongole','takkal','tenali','east-godavari','vijayawada','guntur','east-godavari'],
    'age' : [23,21,21,23,22,23,22],
    'class' : ['cse','cse','cse','cse','cse','ece','ece']
}
df = pd.DataFrame(dic1)
df
```

```
dic2 = {
'name' : ['pramodha','keerthika','chandana','swathi','deepthi','siri','thaheera'],
'dob' : ['07/08/2000','07/07/2002','06/06/2002','22/03/2000','28/10/2000',
'10/07/2000','28/06/2001'],
'place' : ['ongole','takkal','tenali','east-godavari','vijayawada','guntur','east-godavari'],
'age' : [23,21,21,23,22,23,22],
'class' : ['cse','cse','cse','cse','cse','ece','ece']
}
df = pd.DataFrame(dic2)
df
```

EX2:

```
df = pd.read_csv(r'C:\Users\Lenovo\Downloads\datasets\TeluguMovies_dataset.csv')
df
pd.set_option('display.max_rows', None)
df
```

TYPE and SLICING

type(s)	pandas.core.series.Series
type(df)	pandas.core.frame.DataFrame
df[:10]	
df[0:10]	
df[0:10:2]	
df[-10:]	
df[-10:-1]	
df[-1:-10:-2]	

Different ways to create a Series and Dataframe

1. using list
2. using tuple
3. using np array
4. using dictionary
5. using series

SERIES	DATAFRAMES
lst1 = [1,3,5,7,9] s1 = pd.Series(lst1) s1	lst2 = [['pramodha','ongole','cse'], ['keerthika','takkal','cse'], ['chandana','tenali','cse'], ['swathi','east-godavari','cse'], ['deepthi','vijayawada','cse'], ['siri','guntur','ece'], ['thaheera','east-godavari','ece']] df1 = pd.DataFrame(lst2 , index = [313,343,295,296,315,312,447]) df1
tup1 = (1,3,5,7,9) s2 = pd.Series(tup1) s2	tup2 = (('pramodha','ongole','cse'), ('keerthika','takkal','cse'), ('chandana','tenali','cse'), ('swathi','east-godavari','cse'), ('deepthi','vijayawada','cse'), ('siri','guntur','ece'), ('thaheera','east-godavari','ece')) df2 = pd.DataFrame(tup2 , index = (313,343,295,296,315,312,447)) df2
arr1 = np.array(range(1,11,2)) s3 = pd.Series(arr1) s3	arr2 = np.array(['pramodha','keerthika','chandana','swathi', 'deepthi','siri','thaheera']) df3 = pd.DataFrame (arr2) df3

dic1 = {1:1,2:4,3:9,4:16,5:25}
s4 = pd.Series(dic1)
s4
	dic2 = {
	'name' :
	['pramodha','keerthika','chandana','swathi','deepthi','siri','thaheera'],
	'place' : ['ongole','takkal','tenali','east-
	godavari','vijayawada','guntur','east-godavari'],
	'age' : [23,21,21,23,22,23,22],
	'class' : ['cse','cse','cse','cse','cse','ece','ece']
	}
	df4 = pd.DataFrame(dic2)
	df4

	name = ['pramodha','keerthika','chandana','swathi','deepthi','siri','thaheera'] series = pd.Series(name) df5 = pd.DataFrame(series , index = range(0,7)) df5

Creating dataframe using a dictionary of series

```
dic = {

'name':pd.Series(['pramodha','keerthika','chandana','swathi','deepthi','siri','thaheera'],
index = range(1,8)),
    'place' : pd.Series(['ongole','takkal','tenali','east-
godavari','vijayawada','guntur','east-godavari'], index = range(1,8)),
```

```
    'age' : pd.Series([23,21,21,23,22,23,22], index = range(1,8)),
    'class' : pd.Series(['cse','cse','cse','cse','cse','ece','ece'], index = range(1,8))
}
df = pd.DataFrame(dic)
df
```

DataFrame Basic Functionality

help - used to get help related to the object passed during the call	`help(df)`
info - prints information about the DataFrame	`df.info()`

describe - used for calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame	`df.describe ()`
	`df['No.of.Ratings'].describe()`
	`df.describe (include = 'all')`
	`df.describe (include = 'object')`

columns - prints the columns name in the table	`df.columns`
values - returns actual data as ndarray	`df.values`
items - return the list with all dictionary keys with values	`df.items`
transpose - returns transpose of DataFrame	`df.T` `np.transpose(df)`
type - return the type of data stored	`type(df)`
dtype - return datatype of each column	`df.dtypes`
shape - returns tuple representing dimensionallity	`df.shape`
axes - returns list of row axis labels and column axis labels	`df.axes`
head - by default head returns first 5 rows	`df.head()` `df.head(10)`
tail - by default tail returns last 5 rows	`df.tail()` `df.tail(10)`
len - find the number of rows in pandas DataFrame	`len(df)`
unique - used to find the unique values from a series	`df['Movie'].unique()`
value_counts - it will give count each unique data in a given column	`df['Movie'].value_counts()`
set_index - used to set the index to pandas DataFrame	`df = df.set_index('Year')`
sort_index - used to sort the pandas DataFrame by index or columns by name/labels	`df.sort_index()`
sort_values - to sort the DataFrame based on the values in a single column	`df['No.of.Ratings'].sort_values(ascending = False)` `df.sort_values(by = 'No.of.Ratings' , ascending = True)`
nunique - returns the number of unique values for each column	`df['Movie'].nunique()`
isin - checks if the Dataframe contains the specified value(s)	`df['Certificate'].isin(['UA','U'])`
between - used to check if the values of the series object lie in between the boundary values passed to the function	`df['Year'].between(1990,2020)`
replace - replaces the specified value with another specified value	`df['Year'] = df['Year'].replace(np.nan, df['Year'].mean())`

STATISTICAL FUNCTIONS

min - returns the minimum value	`df.min(numeric_only = True)`
	`df['No.of.Ratings'].min()`
	`df[df['No.of.Ratings'] == df['No.of.Ratings'].min()]`

max - returns the maximum value	
sum - returns the sum of values for requested axis, by default axis = 0	`df.sum(numeric_only = True)` `df.sum(numeric_only = True , axis = 0)`
mean - returns the average of values for requested axis, by default axis = 0	
median - returns the average of values excluding outliers for requested axis, by default axis = 0	
mode - returns the most frequently used values for requested axis, by	

default axis = 0
var - returns the varience of values for requested axis, by default axis = 0
std - returns the standard deviation of values for requested axis, by default axis = 0
	`df.sum(numeric_only = True , axis = 1)`
	`df['No.of.Ratings'].sum(numeric_only = True)`
	`df['No.of.Ratings'].sum(numeric_only = True , axis = 0)`
	`df[0:10].sum(numeric_only = True , axis = 0)`
	`df[0:10].sum(numeric_only = True , axis = 1)`

CSV files CREATION and MODIFICATION

Write Dataframe to CSV	`df.to_csv('newfilename.csv')`
Write Dataframe to CSV without index	`df.to_csv('newfilename.csv' , index = False)`
Write Dataframe to XLSX	`df.to_excel('newfilename.xlsx')`
Write Dataframe to XLSX without index	`df.to_excel('newfilename.xlsx' , index = False)`
Write Dataframe to Notepad	`df.to_csv('newfilename.txt')`
Write Dataframe to Notepad without index	`df.to_csv('newfilename.txt' , index = False)`

ACCESSING

```
df
df['Movie']
df[['Movie']]
df[['Movie','No.of.Ratings']]
df['Movie'][0]
df[['Movie']][0:1]
df[['Movie','No.of.Ratings']][0:1]
df['No.of.Ratings'][0:1] > 1000
df['No.of.Ratings'] > 1000
df[['No.of.Ratings']] > 1000
df[df['No.of.Ratings'] > 1000]
```

Position and Label Based Indexing: df.iloc and df.loc

There are two main ways of indexing dataframes:

1. Position based indexing using df.iloc
2. Label based indexing using df.loc

Using both the methods, we will do the following indexing operations on a dataframe:

- Selecting single elements/cells
- Selecting single and multiple rows
- Selecting single and multiple columns
- Selecting multiple rows and columns

Selecting single elements/cells	`df.iloc[1,1]` `df.iloc[[1],[1]]`	`df.loc[1][1]` `df.loc[1 , 'Movie']`

Selecting single and multiple rows	`df.iloc[0]`
	`df.iloc[[0]]`
	`df.iloc[0:10]`
		`df.loc[1]`
		`df.loc[[1]]`
		`df.loc[1:10]`

Selecting single and multiple columns	`df.iloc[:,[1]]`
	`df.iloc[:,[1,2,3,4,5]]`
		`df.loc[:,'Movie']`
		`df.loc[:,['Movie','Overview']]`
		`df.loc[:,'Movie':'Overview']`

Selecting multiple rows and columns	`df.iloc[[1,2,3],[1,2,3]]`	`df.loc[[1,2,3],['Movie','Overview']]`
		`df.loc[[1,30],['Movie','Overview']]`
		`df.loc[1:30,'Movie':'Overview']`

Multi-indexing

It allows us to select more than one row and column in your index. (Multi-indexing = index levels)

```
from numpy.random import randn as rn
g = ['g1','g1','g2','g2','g3','g3']
l = [1,2,3,4,5,6]
indices = list(zip(g,l))
indices
indices = pd.MultiIndex.from_tuples(indices)
indices
df = pd.DataFrame(data = np.round(rn(6,3)), index = indices , columns = ['a1','a2','a3'])
df
df.loc['g1']
df.loc['g1']['a1'][1]
df.loc['g1']['a1'][:]
df.loc['g1'].loc[[1]]
df.loc['g1'].loc[:]
df.loc['g1'][['a1']]
df.loc['g1'][['a1','a2']]
df.loc[['g1','g2']]
df.loc['g1'].loc[1:2,'a1':'a2']
df.loc[['g1','g2']].loc[:,:]
```

Pandas TIME

date_range() - pandas.date_range() is one of the general functions in Pandas which is used to return a fixed frequency DatetimeIndex.	`pd.date_range('2000/07/08' , periods = 5)`
	`pd.date_range('2000/07/08' , periods = 5 , freq = 'Y')`
	`pd.date_range('2000/07/08' , periods = 5 , freq = 'M')`
	`pd.date_range('2000/07/08' , periods = 5 , freq = 'D')`

datetime() - that converts date and time in string format to a DateTime object	`start = pd.datetime(2000,7,8)`
	`start`
	`stop = pd.datetime(2023,7,8)`
	`stop`
	`pd.date_range(start , stop , freq = 'Y')`

to_datetime() - function is used to convert argument to datetime	`df['dob'] = pd.to_datetime(df['dob'])`

`df['dob'].dt.strftime('%m-%Y-%d')`
`df['dob'].dt`
`df['dob'].dt.year`
`df['dob'].dt.month`
`df['dob'].dt.day`
`df['dob'].dt.week`
`df['dob'].dt.day_name()`
`today = pd.to_datetime('today')`
`today`
`today.year`
`today.month`
`today.day`
`today.week`
`today.day_name()`	

Creating, Adding, Dropping and Rearranging Rows and Columns

	Columns	Rows

CREATING	`df['new'] = list(range(1,1401))`	`a = {`
		`'Unnamed: 0' : 1400,`
		`'Movie' : 'Gunturu Karam',`
		`'Year' : 2024,`
		`'Certificate' : 'U',`
		`'Genre' : 'Action',`
		`'Overview' : 'Fighting for the savage of mirchiyard',`
		`'Runtime' : 134,`
		`'Rating' : 9.0,`
		`'No.of.Ratings' : 1372,`
		`}`
		`df = df.append(a , ignore_index = True)`
		`df`

RENAMING	`df.rename(columns = {'new' : 'new column'} , inplace = True)`	`df.iloc[-1,1] = 'Mahesh Gunturu Karam'`
		`df.iloc[-1,1]`
		`df`

REARRANGING	`df.iloc[:,[9,0,1,2,3,4,5,6,7,8]]`
	`df.loc[:,['new column','Unnamed: 0','Movie','Year','Certificate','Genre', 'Overview','Runtime','Rating','No.of.Ratings']]`
		`df1 = df.head(3)`
		`df1`
		`df1.reindex([2,0,1])`

DROPPING	`df.drop(['new column'] , axis = 1 , inplace = True)`	`df.drop([1400] , inplace = True)`

TYPE CASTING and STRING MANIPULATIONS

astype - used for casting the pandas object to a specified dtype	`df['Year'].astype(str)`
strftime - used to convert to Index using specified date_format	`pd.to_datetime(df['Year'])` `pd.to_datetime(df['Year']).dt.strftime('%Y/%d/%m')`
upper - convert DataFrame column values to uppercase	`df['Movie'][2].lower()` `df['Movie'][2].upper()`
lower - convert DataFrame column values to lowercase	`df['Movie'].str.lower()` `df['Movie'].str.upper()`
contains - used to test if pattern or regex is contained within a string of a Series or Index	`df['Certificate'].str.contains('U')`
strip - used to remove leading and trailing characters	`df['Movie'].str.strip()`
split - lets you split a string value up into a list or into separate dataframe columns based on a separator or delimiter value, such as a space or comma	`df['New'] = df['Movie'].str.split(' ')`
expand - one of the window methods of pandas and it provides expanding transformations	`df[['New1','New2']] = df['Certificate'].str.split('U', expand = True)`

GROUPBY , CROSSTAB and PIVOT TABLE

- **groupby** - grouping the data points (i.e. rows) based on the distinct values in the given column or columns
- **crosstab** - one of the many methods that help you reshape your data in Pandas
- **pivot table** - a quantitative table that summarizes a large DataFrame, such as a large dataset

AGGREGATE FUNCTIONS

1. **min** - minimum
2. **max** - maximum
3. **sum** - sum of all items
4. **prod** - product of all items
5. **mean** - average of all items
6. **median** - average of the middle items excluding outliers
7. **var** - varience
8. **std** - standard deviation
9. **count** - count of all items
10. **first** - first item
11. **last** - last item
12. **mad** - mean absolute deviation

ex1 : single grouping

```
df.groupby (by = 'Certificate').min(numeric_only = True)
df.groupby (by = 'Certificate').max(numeric_only = True)
df.groupby (by = 'Certificate').sum(numeric_only = True)
df.groupby (by = 'Certificate').prod(numeric_only = True)
df.groupby (by = 'Certificate').mean(numeric_only = True)
df.groupby (by = 'Certificate').median(numeric_only = True)
df.groupby (by = 'Certificate').var(numeric_only = True)
df.groupby (by = 'Certificate').std(numeric_only = True)
df.groupby (by = 'Certificate').count()
df.groupby (by = 'Certificate').first(numeric_only = True)
df.groupby (by = 'Certificate').last(numeric_only = True)
df.groupby (by = 'Certificate').mad()
```

ex2 : single grouping and single column

```
df.groupby (by = 'Certificate')[['No.of.Ratings']].min(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings']].max(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings']].sum(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings']].prod(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings']].mean(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings']].median(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings']].var(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings']].std(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings']].count()
df.groupby (by = 'Certificate')[['No.of.Ratings']].first(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings']].last(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings']].mad()
```

ex3 : single grouping and multiple columns

```
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].min(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].max(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].sum(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].prod(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].mean(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].median(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].var(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].std(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].count()
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].first(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].last(numeric_only = True)
df.groupby (by = 'Certificate')[['No.of.Ratings','Rating']].mad()
```

ex4 : multiple grouping and single, multiple columns

```
df.groupby (by = ['Certificate','Year']).min(numeric_only = True)
```

```
df.groupby (by = ['Certificate','Year']).max(numeric_only = True)
df.groupby (by = ['Certificate','Year']).sum(numeric_only = True)
df.groupby (by = ['Certificate','Year']).prod(numeric_only = True)
df.groupby (by = ['Certificate','Year'])[['No.of.Ratings']].mean(numeric_only = True)
df.groupby (by = ['Certificate','Year'])[['No.of.Ratings']].median(numeric_only = True)
df.groupby (by = ['Certificate','Year'])[['No.of.Ratings']].var(numeric_only = True)
df.groupby (by = ['Certificate','Year'])[['No.of.Ratings']].std(numeric_only = True)
df.groupby (by = ['Certificate','Year'])[['No.of.Ratings','Rating']].count()
df.groupby (by = ['Certificate','Year'])[['No.of.Ratings','Rating']].first(numeric_only = True)
df.groupby (by = ['Certificate','Year'])[['No.of.Ratings','Rating']].last(numeric_only = True)
df.groupby (by = ['Certificate','Year'])[['No.of.Ratings','Rating']].mad()

>>> df.groupby(by =
['Certificate','Year'])[['No.of.Ratings','Rating']].agg([np.min,np.max,np.sum,np.prod])
>>> pd.crosstab(df['Year'] , df['Rating'])

>>> pd.crosstab(df['Year'] , df['Rating'] , values = df['No.of.Ratings'] , aggfunc = 'mean')
>>> df.pivot_table(values = 'Rating' , index = 'Year' , columns = ['Movie','No.of.Ratings'] , aggfunc
= 'sum')
```

JOINS

concat - to concatenate/merge two or multiple pandas DataFrames across rows or columns	`pd.concat ([df1,df2] , axis = 0)` `pd.concat ([df1,df2] , axis = 1)`
merge - updates the content of two DataFrame by merging them together	`pd.merge(df1,df2)`
inner join - merge two data frames at the intersection	`pd.merge(df1,df2 , how = 'inner' , on = 'name')`
left inner join - includes all records from the left side and matched rows from the right table	`pd.merge(df1,df2 , how = 'left' , on = 'name')`
right inner join - returns all rows from the right side and unmatched rows from the left table	`pd.merge(df1,df2 , how = 'right' , on = 'name')`
outer/full join - returns all rows from both DataFrames	`pd.merge(df1,df2 , how = 'outer' , on = 'name')`
cartesian join - create the cartesian product of rows of both frames	`pd.merge(df1,df2 , how = 'cross')`

SPECIAL FUNCTIONS

query - takes a query expression as a string parameter, which has to evaluate to either True of False	`df.query("Movie == '1 - Nenokkadine' and Certificate != 'U'")`
nlargest - return the first n rows in descending order, with the largest values in columns	`df.nlargest(5,'Rating')`
nsmallest - return the first n rows in ascending order, with the smallest values in columns	`df.nsmallest(5,'Rating')`

copy - returns a copy of the DataFrame	`dfnew = df.copy()`
	`dfnew.drop(['Year'] , axis = 1 , inplace = True)`
	`dfnew`
	`df`

map - map the values of a series to another set of values or run a custom function	`new = {'UA':1, 'U':0}`
	`df['CertificateMap'] = df['Certificate'].map(new)`
	`df`

apply - used to apply a function along an axis of the DataFrame	`def newera(x) :`
	`  if x>=2000 :`
	`  return '21st Gen'`
	`  else :`
	`  return '20th Gen'`
	`df['NewEraApply'] = df['Year'].apply(newera)`
	`df`

lambda - a small anonymous function that can take any number of arguments and execute an expression	`df['NewEraLambda'] = df['Year'].apply(lambda x : '21st Gen' if x>=2000 else '20th Gen')`

transform - used to call function on self producing a Series with transformed values and that has the same axis length as self	dic ={
	'name' :
	['pramodha','keerthika','chandana','swathi','deepthi',
	'siri','thaheera'],
	'dob' :
	['07/08/2000','07/07/2002','06/06/2002','22/03/2000',
	'28/10/2000','10/07/2000','28/06/2001'],
	'place' : ['ongole','takkal','tenali','east-
	godavari','vijayawada','guntur','east-godavari'],
	'age' : [23,21,21,23,22,23,22],
	'class' : ['cse','cse','cse','cse','cse','ece','ece']
	}
	df = pd.DataFrame(dic)
	df['age'].transform(**lambda** x : x+1)

filter - filters the DataFrame, and returns only the rows or columns that are specified in the filter	df.filter(items = ['name' , 'class'])
iterrows - iterate over DataFrame rows as (index, Series) pairs	next(df.iterrows())[1]
stack - used to reshape the given DataFrame by transposing specified column level into row level **unstack** - reshape the given Pandas DataFrame by transposing specified row level to column level	**from** numpy.random **import** randn **as** rn g = ['g1','g1','g2','g2','g3','g3'] l = [1,2,3,4,5,6] indices = list(zip(g,l)) indices indices = pd.MultiIndex.from_tuples(indices) indices df = pd.DataFrame(data = np.round(rn(6,3)), index = indices , columns = ['a1','a2','a3']) df df.stack(0) df.unstack(0)

melt - enables us to reshape and elongate the data frames in a user-defined manner	dic ={
	'name' :
	['pramodha','keerthika','chandana','swathi','deepthi',
	'siri','thaheera'],
	'dob' :
	['07/08/2000','07/07/2002','06/06/2002','22/03/2000',
	'28/10/2000','10/07/2000','28/06/2001'],
	'place' : ['ongole','takkal','tenali','east-
	godavari','vijayawada','guntur','east-godavari'],
	'age' : [23,21,21,23,22,23,22],
	'class' : ['cse','cse','cse','cse','cse','ece','ece']
	}
	df = pd.DataFrame(dic)
	df.melt(id_vars = ['name'] , value_vars = ['class'])