Short Notes

__/__/____

Pandas

- Data Structures in pandas :
  - ↳ Series (similar to 1 dimensional numpy array)
  - ↳ Data Frame (Similar to 2 dim. numpy array)

Series

S = pd.Series(data, index = idx)

list → data = [1, 2, 3, 4]

dictionary → data = {'a' : 1, 'b' : 2}

numpy array → data = np.array([1, 2, 3])

scalar → data = 5, index = ['a', 'b', 'c']

properties / Attributes

- Data types : S.dtype
- Shape : S.shape
- Values : S.values
- Array : S.array

Methods

- Head : S.head()
- Tail : S.tail()
- Info : S.info()
- To numpy : S.to_numpy()

Indexing

- Single Index
  - S[2] ← using Index value
  - S['a'] ← using Row label
- Multiple Indexes
  - S[[1, 4]] (OR) S[['a', 'c']]
    - ↳ list of indexes

slicing

S[start : stop : step]

__/__/____

Data Frame

df = pd.DataFrame(data, index = idxs, columns = cols)

create data frame :

using dictionary
```
data = {'Name' : ['Ravi', 'Shaheer'], 'Age' : [26, 23]}
```

using tuple
```
data = (('Ravi', 26), ('Shaheer', 23)),
columns = ['Name', 'age'].
```

using list
```
data = (['Ravi', 26], ['Shaheer', 23]), columns = ['Name', 'Age']
```

using numpy array
```
data = np.array([1, 2, 3], [4, 5, 6]), columns = [col_1, col_2]
```

- Attributes / properties

- Shape of df	: df.shape
- columns	: df.columns
- data types	: df.dtypes
- Axes	: df.axes
- values	: df.values

Methods

- Head	: df.head()
- Tail	: df.tail()
- Info	: df.info()

- Load data to pandas :
- pd.read_* (* = csv, excel, sql, json, etc.....)

Eg: pd.read_csv('path')

- Export / save from pandas :
- df.to_* (* = csv, excel, sql, json, etc.....)

Ex: df.to_excel('path', sheet_name = 'name', index = False)

__/__/____

Non-Visual Data Analysis (statistical)

- min `df.min()`
- max `df.max()`
- mean `df.mean()`
- median `df.median()`
- var `df.var()`
- std `df.std()`
- skew `df.skew()`
- kurt `df.kurt()`
- sum `df.sum()`
- count `df.count()`

- count `df.count()`
- nunique `df.nunique()`
- unique `df.unique()`
- value_counts `df.value_counts()`

- Describe → `df.describe(include = ['object'])`
  `include = ['number']`
  `include = 'all'`

- `num_df = df.select_dtypes(include = ['int32', 'float64'])`
- `cat_df = df.select_dtypes(include = ['object'])`

- Aggregation : `df['col-1'].agg(['min', 'max', 'mean', 'median', 'std'])`

`df.agg({'col-1': ['min', 'max', 'mean'], 'col-2': ['count', 'unique']})`

- Indexing & Slicing
- Filtering Single column vs Multiple columns :

- Single column : `df['col-name']` (OR) `df[['col-name']]`

- multiple columns : `df[['col-1', 'col-2', 'col-3']]`

__/__/____

- Filtering Rows :-
- Way-1 : df [starting_row_index : ending_row_index : step]
- Way-2 : df [condition]

```
df [df['date'].isin(['10-5-2016', '10-4-2016'])
```

- Filtering Specific Rows & columns from df :

```
df.loc[row_label, column_label]
df.iloc[row_index, column_index]
```

- Accessing Rows based on condition :

```
df.loc[condition , column_labels]
```

- Accessing Rows based on multiple condition :

```
df.loc[(cond-1) &(cond-2) | (cond-3), column_labels]
```

- Renaming columns :

```
df.rename(index = None, columns = {'old_name': 'New_name'})
```

- Modify columns datatype :
- .astype( )

```
df['quantity'] = df['quantity'].astype(int)
df = df.astype({'quantity': int})
```

- .apply( )

```
df['quantity'] = df['quantity'].apply(pd.to_numeric
                                      pd.to_datetime
                                      pd.to_timedelta.
```

/ /

- creating a Derived column :

```
df['amount'] = df['quantity'] * df['unit_price']
```

- creating columns using apply() function :

```
df.apply(function, axis= )
s.apply(function, axis= )

df['amount'] = df.apply(lambda row: row['quantity'] *
                        row['unit_price'], axis=1)
```

date time

properties		methods	
- day	df['date'].dt.day	- day_name	.dt.day_name()
- month	.dt.month	- month_name	.dt.month_name()
- years	.dt.years		
- day_of_week	.dt.day_of_week		
- day_of_years	.dt.day_of_years		
- is_leap_years	.dt.is_leap_years		
- week	.dt.week		

time delta

properties		methods	
- days	.dt.days	- total seconds	.dt.total_seconds()
- hours	.dt.hours		
- minutes	.dt.minutes		
- Seconds	.dt.seconds		
- Components	.dt.components		

__/__/____

- pd.to_datetime( )

```
df['date'] = pd.to_datetime(df['date'], dayfirst = True
                                        format = '%d/%m/%Y')

df = pd.read_csv('path', parse_dates = ['cols'])
```

- Sort_index, Sort_values, Reset_index :

```
df.sort_index(ascending = True)
df.sort_values(by = 'col_name')
df.reset_index( )
```

Split :

```
grouped_df = df.groupby('category')
grouped_df = df.groupby(['category', 'sub_category'])
```

- Groups : grouped_df.groups
  grouped_df.groups.keys( )
- filter : grouped_df.get_group('Group_key_name')
  grouped_df.get_group(('Group_category', 'Group_sub_cat'))
- first, last & nth row of each group :
  grouped_df.first( )
  grouped_df.last( )
  grouped_df.nth( )

Apply :

- Aggregation :
  cov( ) } Built-in
  idxmax( ) }
  grouped_df['sales'].agg(['min', 'max']) } user defined

__/__/____

- Filteration:

head ( )
tail ( )
nth ( )
} Built-in

user defined { grouped_df.filter(lambda grp : grp['sales'].mean >20)

- Transformation:

bfill ( )
cumsum ( )
} Built-in

grouped_df.transform(lambda x : x+1) } user defined.

- pivot_table :

df.pivot_table(values ='sales', index =['region'],
agg_func =['sum'], columns =['category'])

- pd.merge(=''):

pd.merge(left, right, how ='inner', on=None, left_on = None,
right_on = None, left_index=None, right_index= ,
sort =)

how = 'inner'
= 'outer'
= 'left'
= 'right'
= 'cross'

__/__/____

- plotting :

```
df.plot(kind='kind')
```

kind = 'bar'
= 'barh'
= 'hist'
= 'kde'

- Delete columns :

Syntax-1 `df.drop(['col1', 'col3'], axis=1, inplace=True)`

Syntax-2 `df.drop(df.loc[:, 'col2':'col4'], inplace=True)`

Syntax-3 `df.drop(df.columns[[0,4,2]], axis=1, inplace=True)`

Syntax-4 `df.drop(df.iloc[:, 1:3], inplace=True)`

Syntax-5 `df.pop('col4', inplace=True)`

- Inserting Rows :

```
df = pd.concat([df, new_df], ignore_index=True, axis=0)
```

- Inserting a row using df.loc[ ]

```
df.loc[len(df)] = ['1/12/2017', 28, 2, 'Rain']
```

- Inserting a row at Specific index

```
df.loc[8.5] = ['1/11/2017', 30, 3, 'Rain']
```

↳ sort & Reset index

```
df = df.sort_index().reset_index(drop=True)
```