Series

```
myindex = ['USA','Canada','Mexico']
mydata = [1776,1867,1821]
pd.Series(data=mydata,index=myindex) #creating series from list

ran_data = np.random.randint(0,100,4)
names = ['Andrew','Bobo','Claire','David']
ages = pd.Series(data=ran_data,index=names) #creating series from numpy
array

mydata = {'Sammy':5,'Frank':10,'Spike':7}
ages = pd.Series(mydata) #creating series from dictionary
```

DataFrames

```
#creating a dataframe
mydata = np.random.randint(0,101,(4,3))
myindex = ['CA','NY','AZ','TX']
mycolumns = ['Jan','Feb','Mar']
df = pd.DataFrame(data=mydata,index=myindex,columns=mycolumns)

#reading a csv file
df = pd.read_csv(filepath)

#obtaining basic information about dataframe
df.columns, df.index, df.head(5),df.sample(5),df.tail(5)
df.info(),len(df),df.describe(), df.describe().transpose()
type(df['column_name'])


#Selection and Indexing
*columns
df['column_name'] #grab single column
df[['column_1','column_2']] #grab multiple column
#creating new column
df['tip_percentage'] = 100* df['tip'] / df['total_bill']
#adjusting existing column
df['price_per_person'] = np.round(df['price_per_person'],2)
#removing column
df = df.drop("tip_percentage",axis=1)

*Index Basics
df = df.set_index('Payment ID')
df = df.reset_index()
```

```
*Rows
df = df.set_index('Payment ID')
#grab single row
df.iloc[0] #integer based
df.loc['Sun2959'] #name based
df.iloc[0:4] #grab multiple row
df.loc[['Sun2959','Sun5260']] #grab multiple row
df = df.drop(0,axis=0) #removing a row
one_row = df.iloc[0]
df = df.append(one_row) #adding a row

Conditional Filtering

# df['total_bill'] > 30
bool_series = df['total_bill'] > 30
df[bool_series]
df[df['total_bill']>30], df[df['sex'] == 'Male']
df[(df['total_bill'] > 30) & (df['sex']=='Male')]
df[(df['total_bill'] > 30) & (df['sex']!='Male')]
df[(df['day'] =='Sun') | (df['day']=='Sat')]

options = ['Sat','Sun']
df['day'].isin(options) #Conditional Operator isin()
df[df['day'].isin(['Sat','Sun'])]


Useful Method
def last_four(num):
    return str(num)[-4:]
df['last_four'] = df['CC Number'].apply(last_four)

def yelp(price):
    if price < 10:
        return '$'
    elif price >= 10 and price < 30:
        return '$$'
    else:
        return '$$$'
df['Expensive'] = df['total_bill'].apply(yelp)

#apply with lambda
df['total_bill'].apply(lambda bill:bill*0.18)
#apply that uses multiple columns
def quality(total_bill,tip):
    if tip/total_bill  > 0.25:
```

```
        return "Generous"
    else:
        return "Other"
df['Tip Quality'] = df[['total_bill','tip']].apply(lambda df:
quality(df['total_bill'],df['tip']),axis=1)
df['Tip Quality'] = np.vectorize(quality)(df['total_bill'], df['tip'])

df.sort_values('tip') #sorting column values
df.sort_values(['tip','size'])

df.corr()
df[['total_bill','tip']].corr()

df['total_bill'].max(), df['total_bill'].idxmax(),
df['total_bill'].idxmin()

df['sex'].value_counts()

df['Tip Quality'] = df['Tip
Quality'].replace(to_replace='Other',value='Ok')

df['size'].unique(), df['size'].nunique()

my_map = {'Dinner':'D','Lunch':'L'}
df['time'].map(my_map)

simple_df.duplicated(), simple_df.drop_duplicates()

df['total_bill'].between(10,20,inclusive=True)
df.nlargest(10,'tip'), df.nsmallest(10,'tip')


Missing Value

#null value
df.isnull(), df.isnull().sum(), df.notnull()
df[df['first_name'].notnull()]
df[(df['pre_movie_score'].isnull()) & df['sex'].notnull()]
df.dropna(), df.dropna(thresh=1)
#filling null value
df.fillna("NEW VALUE!")
df['first_name'] = df['first_name'].fillna("Empty")
df['pre_movie_score']=df['pre_movie_score'].fillna(df['pre_movie_score'].me
an())
```

Groupby Operations and Multi-level Index

```
# Creates a groupby object waiting for an aggregate method
df.groupby('model_year')
# model_year becomes the index! It is NOT a column name,it is now the name
of the index
df.groupby('model_year').mean()

avg_year = df.groupby('model_year').mean()
avg_year.index
Int64Index([70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82],
dtype='int64', name='model_year')
avg_year.columns
Index(['mpg', 'cylinders', 'displacement', 'weight', 'acceleration',
'origin'], dtype='object')
avg_year['mpg']

df.groupby('model_year').describe()
df.groupby('model_year').describe().transpose()

#groupby multiple columns
df.groupby(['model_year','cylinders']).mean()

df.groupby(['model_year','cylinders']).mean().index
MultiIndex([(70, 4),(70, 6),...])


#Indexing with the Hierarchical Index
year_cyl = df.groupby(['model_year','cylinders']).mean()
year_cyl.index.levels
FrozenList([[70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82], [3, 4, 5,
6, 8]])
year_cyl.index.names
FrozenList(['model_year', 'cylinders'])
year_cyl.loc[70], year_cyl.loc[[70,72]], year_cyl.loc[(70,8)]

#Grab Based on Cross-section with .xs()
year_cyl.xs(key=70,axis=0,level='model_year')
year_cyl.xs(key=4,axis=0,level='cylinders')

df[df['cylinders'].isin([6,8])].groupby(['model_year','cylinders']).mean()

#Sorting MultiIndex
year_cyl.sort_index(level='model_year',ascending=False)
year_cyl.sort_index(level='cylinders',ascending=False)
```

```
#agg() on a DataFrame
df.agg(['median','mean'])
df.agg(['sum','mean'])[['mpg','weight']]

#Specify aggregate methods per column
df.agg({'mpg':['median','mean'],'weight':['mean','std']})

#agg() with groupby()
df.groupby('model_year').agg({'mpg':['median','mean'],'weight':['mean','std
']})
```

Combining DataFrames

```
<----------Concatenation---------->
data_one = {'A': ['A0', 'A1', 'A2', 'A3'],'B': ['B0', 'B1', 'B2', 'B3']}
data_two = {'C': ['C0', 'C1', 'C2', 'C3'], 'D': ['D0', 'D1', 'D2', 'D3']}
one = pd.DataFrame(data_one)
two = pd.DataFrame(data_two)

#Concatenate along rows
axis0 = pd.concat([one,two],axis=0)
#Concatenate along columns
axis1 = pd.concat([one,two],axis=1)

<----------Merge---------->

registrations =
pd.DataFrame({'reg_id':[1,2,3,4],'name':['Andrew','Bobo','Claire','David']}
)
logins =
pd.DataFrame({'log_id':[1,2,3,4],'name':['Xavier','Andrew','Yolanda','Bobo'
]})
```

```
registrations              |    logins
        reg_id  name       |           log_id   name
0       1       Andrew     |   0        1       Xavier
1       2       Bobo       |   1        2       Andrew
2       3       claire     |   2        3       Yolanda
3       4       David      |   3        4       Bobo
```

```
#inner join
pd.merge(registrations,logins,how='inner',on='name')
```

```
        reg_id  name    log_id
0       1       Andrew  2
1       2       Bobo    4
```

```
#left join
pd.merge(registrations,logins,how='left')
```

```
        reg_id  name    log_id
0       1       Andrew  2.0
1       2       Bobo    4.0
2       3       Claire  NaN
```

```
3       4       David   NaN


#right join
pd.merge(registrations,logins,how='right')

        reg_id  name    log_id
0       1.0     Andrew  2
1       2.0     Bobo    4
2       NaN     Xavier  1
3       NaN     Yolanda 3

#outer join
pd.merge(registrations,logins,how='outer')

        reg_id  name    log_id
0       1.0     Andrew  2.0
1       2.0     Bobo    4.0
2       3.0     Claire  NaN
3       4.0     David   NaN
4       NaN     Xavier  1.0
5       NaN     Yolanda 3.0
```

```
<--Text Methods on Pandas String Column-->

names = pd.Series(['andrew','bobo','claire','david','4'])
names.str.capitalize()
names.str.isdigit()

messy_names = pd.Series(["andrew  ","bo;bo","  claire  "])
messy_names.str.replace(";","")
messy_names.str.replace(";","").str.strip().str.capitalize()

# Alternative with Custom apply() call
def cleanup(name):
    name = name.replace(";","")
    name = name.strip()
    name = name.capitalize()
    return namemessy_names.apply(cleanup)
messy_names.apply(cleanup)

<----Time Methods------->
#Converting to datetime
obvi_euro_date = '31-12-2000' #(dd-mm-yyyy)
pd.to_datetime(obvi_euro_date) #Timestamp('2000-12-31 00:00:00')
#default(yyyy-mm-dd)

# 10th of Dec OR 12th of October?
euro_date = '10-12-2000' #(dd-mm-yyyy)
pd.to_datetime(euro_date) #Timestamp('2000-10-12 00:00:00')
pd.to_datetime(euro_date,dayfirst=True) #Timestamp('2000-12-10 00:00:00')

style_date = '12--Dec--2000'
pd.to_datetime(style_date, format='%d--%b--%Y') #Timestamp('2000-12-12
00:00:00')

sales = pd.read_csv('RetailSales_BeerWineLiquor.csv')
sales['DATE'] = pd.to_datetime(sales['DATE'])

sales = sales.set_index("DATE")
# Yearly Means
sales.resample(rule='A').mean()
https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#off
set-aliases
#help(sales['DATE'].dt)
sales['DATE'].dt.month
sales['DATE'].dt.is_leap_year

<----Inputs and Outputs---->
```

```
#reading a csv file
df = pd.read_csv("C:\\Users\\myself\\files\\some_file.csv")
#saving df in csv file
df.to_csv('new_file.csv',index=False)

df = pd.read_excel('my_excel_file.xlsx',sheet_name='First_Sheet')
# Returns a list of sheet_names
pd.ExcelFile('my_excel_file.xlsx').sheet_names
#grab all sheets
excel_sheets = pd.read_excel('my_excel_file.xlsx',sheet_name=None)
type(excel_sheets) #dict
excel_sheets.keys() #dict_keys(['First_Sheet'])
excel_sheets['First_Sheet']
df.to_excel('example.xlsx',sheet_name='First_Sheet',index=False)

<-----Pivot Tables-------->

df = pd.read_csv('Sales_Funnel_CRM.csv')
pd.pivot_table(df,index="Company",aggfunc='sum')[['Licenses','Sale Price']]
pd.pivot_table(df,index="Company",aggfunc='sum',values=['Licenses','Sale
Price'])
pd.pivot_table(df,index=["Account Manager","Contact"],values=['Sale
Price'],aggfunc='sum')
pd.pivot_table(df,index=["Account Manager","Contact"],values=["Sale
Price"],columns=["Product"],aggfunc=[np.sum])
pd.pivot_table(df,index=["Account
Manager","Contact","Product"],values=["Sale Price","Licenses"],
               aggfunc=[np.sum],fill_value=0,margins=True)
```