Pandas Workshop

In [32]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
% matplotlib inline
```

In [33]:

```
pd.__version__
```

Out[33]:

```
'0.23.0'
```

Pandas Series

Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.).

In [34]:

```
#Creating Pandas Series using lists

animals = ['Tiger', 'Bear', 'Lion']
pd.Series(animals)
```

Out[34]:

```
0    Tiger
1     Bear
2     Lion
dtype: object
```

In [35]:

```
numbers = [1, 2, 3]
pd.Series(numbers)
```

Out[35]:

```
0    1
1    2
2    3
dtype: int64
```

In [36]:

```
animals = ['Tiger', 'Bear', None]
pd.Series(animals)
```

Out[36]:

```
0    Tiger
1     Bear
2     None
dtype: object
```

In [37]:

```
numbers = [1, 2, None]
pd.Series(numbers)
```

Out[37]:

```
0    1.0
1    2.0
2    NaN
dtype: float64
```

In [38]:

```
#Index of a Series can be of String type

sportsDict = {'Archery': 'Bhutan',
              'Golf': 'Scotland',
              'Sumo': 'Japan',
              'Hockey': 'India'}          #Creating Series from Dictionary

sports = pd.Series(sportsDict)
sports
```

Out[38]:

```
Archery      Bhutan
Golf       Scotland
Sumo          Japan
Hockey        India
dtype: object
```

In [39]:

```
#Prints only indices and index type of a series

sports.index
```

Out[39]:

```
Index(['Archery', 'Golf', 'Sumo', 'Hockey'], dtype='object')
```

In [40]:

```
#Another way of creating a series

nations = ['India', 'America', 'Sri Lanka']
nationalAnimals = pd.Series(animals, index = nations)
nationalAnimals
```

Out[40]:

```
India        Tiger
America       Bear
Sri Lanka     None
dtype: object
```

Querying a Series

In [41]:

```
sectionNames = ['A','B','C','D','E','F']
```

In [42]:

```
sectionStrength = [34,50,60,54,45,40]
```

In [43]:

```
sections = pd.Series(sectionStrength,index = sectionNames)
```

In [44]:

```
sections.head()
```

Out[44]:

```
A    34
B    50
C    60
D    54
E    45
dtype: int64
```

In [45]:

```
sections.tail(3)
```

Out[45]:

```
D    54
E    45
F    40
dtype: int64
```

In [46]:

```
#Querying a Series based on index location

sections.iloc[4]
```

Out[46]:

```
45
```

In [47]:

```
#Querying a Series based on index label

sections.loc['B']
```

Out[47]:

```
50
```

In [48]:

```
#This type of indexing uses the index labels by default to query Series if all index label
#index positons to query. For above series it will use index positons.

sections[3]     #Avoid this type of indexing since it can be misleading in case of numeric
```

Out[48]:

```
54
```

In [49]:

```
countryCodeDict ={91: 'India',
                  1: 'America',
                 20: 'Egypt',
                 33: 'France'}
countryCode = pd.Series(countryCodeDict)
countryCode
```

Out[49]:

```
91      India
1     America
20      Egypt
33     France
dtype: object
```

In [50]:

```
 #using index labels
countryCode[20]
```

Out[50]:

```
'Egypt'
```

In [51]:

```
def invertSeries(s):
    '''
    objective:
        to ivert the role of indexes and values in a series
    input:
        s: series
    output:
        invertedSeries: inverted series s
    '''
    '''
        approach:
            initialize an empty series invertedSeries
            iterate through iteritems to append iverted entries from the series s to invert
    '''
    invertedSeries = pd.Series()
    for label, value in s.iteritems():
        invertedSeries.at[value] = label
    return invertedSeries
```

In [52]:

```
invertedSeries = invertSeries(countryCode)
invertedSeries
```

Out[52]:

```
India      91
America     1
Egypt      20
France     33
dtype: int64
```

Operations on a Series

In [53]:

```
#Adding values of a series  using for loop:

def getClassStrength(sections):
    '''
    objective: to compute the strength of the class across the sections

    input:
        sections: series comprising sections of a class
    output:
        classStrength: total number of students across the sections
    '''
    '''
    approach:
        initialize classStrength = 0
        iterate through iteritems
    '''
    classStrength = 0
    for item in sections:
        classStrength += item
    return classStrength
```

In [54]:

```
getClassStrength(sections)
```

Out[54]:

```
283
```

In [55]:

```
#Add series Elements using numpy function sum() - Using numpy functions is much faster than

np.sum(sections)
```

Out[55]:

```
283
```

In [31]:

```
#To see the performance difference between for loop and numpy sum() let us create a big ser

randomNos = pd.Series(np.random.randint(0,1000,10000))
randomNos.head()
```

Out[31]:

```
0    424
1    101
2    627
3    394
4    670
dtype: int32
```

In [56]:

```
len(randomNos)
```

Out[56]:

```
10000
```

In [57]:

```
%%timeit -n 100
total = 0
for item in randomNos:
    total += item
```

```
1.7 ms ± 265 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [58]:

```
%%timeit -n 100
total = np.sum(randomNos)
```

```
164 µs ± 23.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [59]:

```
#Iterating and Operating over every item of a series

def increaseStrength(sections, increment):
    '''
    objective:
        to increase strength of each section by increment
    inputs:
        sections: input series
        increment: value by which saction strength is to be incremented
    output:
        sections: each senction strength incremented by increment
    '''

    #approach: iterate through iteritems

    for label, value in sections.iteritems():
        sections.at[label] = value + increment
    return sections
```

In [60]:

```
sections = increaseStrength(sections, 5)
sections.head()
```

Out[60]:

```
A    39
B    55
C    65
D    59
E    50
dtype: int64
```

In [61]:

```
#adds 5 to each value in Class using broadcasting - It is much faster

sections += 5
sections
```

Out[61]:

```
A    44
B    60
C    70
D    64
E    55
F    50
dtype: int64
```

In [63]:

```
#Appending a Series to another Series

nationalSports = pd.Series({ 'Archery': 'Bhutan',
                             'Golf': 'Scotland',
                             'Sumo': 'Japan',
                             'Taekwondo': 'South Korea'})
cricketLovingCountries = pd.Series(['Australia',
                                    'Barbados',
                                    'Pakistan',
                                    'England'],
                                   index = ['Cricket',
                                            'Cricket',
                                            'Cricket',
                                            'Cricket'])
allCountries = nationalSports.append(cricketLovingCountries)
```

In [64]:

```
nationalSports
```

Out[64]:

```
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
dtype: object
```

In [65]:

```
cricketLovingCountries
```

Out[65]:

```
Cricket    Australia
Cricket     Barbados
Cricket     Pakistan
Cricket      England
dtype: object
```

In [66]:

```
allCountries
```

Out[66]:

```
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
Cricket        Australia
Cricket         Barbados
Cricket         Pakistan
Cricket          England
dtype: object
```

In [67]:

```
allCountries.loc['Cricket']
```

Out[67]:

```
Cricket    Australia
Cricket     Barbados
Cricket     Pakistan
Cricket      England
dtype: object
```

Pandas Dataframe

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects.

In [68]:

```
#Creating a Dataframe from multiple Series: Series indices are promoted to column headers

import pandas as pd
purchase_1 = pd.Series({'Name': 'Ashish',
                        'Item Purchased': 'Bread',
                        'Cost': 22.50})
purchase_2 = pd.Series({'Name': 'Nikita',
                        'Item Purchased': 'Vegetables',
                        'Cost': 90.00})
purchase_3 = pd.Series({'Name': 'Vinod',
                        'Item Purchased': 'Milk',
                        'Cost': 75.00})
df = pd.DataFrame([purchase_1, purchase_2, purchase_3], index = [1, 2, 3])
purchase_1
```

Out[68]:

```
Name              Ashish
Item Purchased     Bread
Cost                22.5
dtype: object
```

In [69]:

```
df.head()
```

Out[69]:

	Name	Item Purchased	Cost
1	Ashish	Bread	22.5
2	Nikita	Vegetables	90.0
3	Vinod	Milk	75.0

In [70]:

```
#Selecting a Column from a Dataframe
Names = df['Name']
print(Names)
```

```
1    Ashish
2    Nikita
3     Vinod
Name: Name, dtype: object
```

Dataframe Operations

In [71]:

```
#Reading a CSV file into a Dataframe
shopping = pd.read_csv("Grocery.csv")
```

In [72]:

```
shopping
```

Out[72]:

	Product	Category	Price	Quantity
0	Bread	Food	20	2
1	Milk	Food	60	5
2	Biscuit	Food	20	2
3	Bourn-Vita	Food	70	1
4	Maggi	Food	20	5
5	Tea	Food	120	1
6	Soap	Hygiene	40	4
7	Brush	Hygiene	30	2
8	Detergent	Household	80	1
9	Hair-Oil	Hygiene	100	1
10	Perfume	Hygiene	150	1
11	Tiffin Box	Household	75	2
12	Pen	Stationary	5	10
13	Pencil	Stationary	2	10
14	T-Shirt	Clothes	250	3
15	Bottle	Household	80	2
16	Bucket	Household	200	1
17	Chips	Food	10	15
18	Juice	Food	100	4
19	Tissues	Hygiene	30	5

In [73]:

```
# Multiplying two Columns to get Total Price for an item
Total = shopping['Price']*shopping['Quantity']
# Adding a new Column to the Dataframe
shopping['Total Price'] = Total
```

In [74]:

```
shopping['Total Price'] = shopping['Price']*shopping['Quantity']
```

In [75]:

```
shopping
```

Out[75]:

	Product	Category	Price	Quantity	Total Price
0	Bread	Food	20	2	40
1	Milk	Food	60	5	300
2	Biscuit	Food	20	2	40
3	Bourn-Vita	Food	70	1	70
4	Maggi	Food	20	5	100
5	Tea	Food	120	1	120
6	Soap	Hygiene	40	4	160
7	Brush	Hygiene	30	2	60
8	Detergent	Household	80	1	80
9	Hair-Oil	Hygiene	100	1	100
10	Perfume	Hygiene	150	1	150
11	Tiffin Box	Household	75	2	150
12	Pen	Stationary	5	10	50
13	Pencil	Stationary	2	10	20
14	T-Shirt	Clothes	250	3	750
15	Bottle	Household	80	2	160
16	Bucket	Household	200	1	200
17	Chips	Food	10	15	150
18	Juice	Food	100	4	400
19	Tissues	Hygiene	30	5	150

In [76]:

```
#Adding Values in Total Price Columm to get total expenditure
totalExpenditure = shopping['Total Price'].sum()
totalExpenditure
```

Out[76]:

```
3250
```

In [ ]:

```
#Sorting Data in decreasing order of item price
shoppingSorted = shopping.sort_values('Price',ascending = False)
shoppingSorted
```

In [ ]:

```
shopping[['Price','Quantity','Total Price']].agg(['sum','mean','max','min','count','median'
```

In [ ]:

```
shopping.describe()
```

In [ ]:

```
shopping.quantile([0.25,0.5,0.75])
```

In [ ]:

```
#Boolean Masking in Pandas
#Mask for products having price greater than 75 percentile of price

Mask = shopping['Price']>shopping['Price'].quantile(0.75)
```

In [ ]:

```
shopping[Mask]
```

In [ ]:

```
#Groupby in Pandas
#All products grouped according to the category
Categories = shopping.groupby('Category')
#Total expenditure per category
Categories['Total Price'].sum()
```

In [ ]:

```
#Total counts per category
Categories['Total Price'].count()
```

In [ ]:

```
#Broadcasting in Pandas Dataframe
 #Increase the quantity of each product by 1
shopping['Quantity'] += 1
```

In [ ]:

```
shopping
```

In [ ]:

```
#Pivoting : Reshapes the dataframe according to the given column,index and values

shoppingPivot = shopping.pivot(index = 'Category',columns = 'Product',values = 'Price')
shoppingPivot
```

In [ ]:

```
#Filling NaN values with Not Applicable

shoppingPivot.fillna("NA",inplace=True)
```

In [ ]:

```
shoppingPivot
```

In [ ]:

```
#deleting columns from a Dataframe

shoppingPivot.drop(['Bourn-Vita','Chips'],axis = 1)
```

In [ ]:

```
#Combining data frames

shoppingNew = pd.read_csv('Grocery2.csv')
shoppingNew
```

In [ ]:

```
shoppingFinal = pd.concat([shopping,shoppingNew])
```

In [ ]:

```
shoppingFinal
```

In [ ]:

```
#Resetting the row index

shoppingFinal = shoppingFinal.reset_index(drop = True)
shoppingFinal
```

In [ ]:

```
#Altering the labels/Column headers

shoppingFinal = shoppingFinal.rename(columns = {'Product': 'Item'})
shoppingFinal
```

In [ ]:

```
#Reindexing the columns

shoppingFinal.reindex(columns = ['Item','Category','Price','Quantity','Total Price'])
```

Data Plotting in Pandas

In [ ]:

```
globalTemp = pd.read_csv('globalTemp.csv')
```

In [ ]:

```
globalTemp.head()
```

In [ ]:

```
#Histogram

hist = globalTemp['LandAverageTemperature'].plot(kind = 'hist',figsize = (12, 8))
hist.set_xlabel("Average Land Temperature")
hist.set_ylabel("Frequency")
```

In [ ]:

```
#Boxplot

box = globalTemp.plot(kind = 'box',figsize = (12, 8))
box.set_ylabel("Temperature")
```

In [ ]:

```
populationData = pd.read_csv('Demographicdata.csv',index_col = 0)
```

In [ ]:

```
populationData.head()
```

In [ ]:

```
#Bar Graph

populationData['Population(in millions)'].plot(kind = 'bar', figsize = (12, 6), title = 'Po
```

In [ ]:

```
# Scatter plot

plt.figure(figsize = (15, 5))
plt.scatter(x = populationData['Total Area(in miles)'], y = populationData['Population(in m
plt.title("Population Scatter")
plt.xlabel("Total Area (in miles)")
plt.ylabel("Population (in millions)")
plt.show()
```

In [ ]:

```
#Linear Regression plot

fig, axis = plt.subplots()
fit = np.polyfit(populationData['Total Area(in miles)'], populationData['Population(in mill
axis.plot(populationData['Total Area(in miles)'], fit[0] * populationData['Total Area(in mi
axis.scatter(populationData['Total Area(in miles)'], populationData['Population(in millions
```

Handling Missing Values

In [ ]:

```
#Handling Missing Values

globalTemp.head()
```

In [ ]:

```
#Displaying rows with missing values

globalTemp[np.isnan(globalTemp['LandAverageTemperature'])]
```

In [ ]:

```
#Filling Missing values with the mean values of the column

globalTemp['LandAverageTemperature'] = globalTemp['LandAverageTemperature'].fillna(globalTe
```

In [ ]:

```
#Now there are no Mising Values

globalTemp[np.isnan(globalTemp['LandAverageTemperature'])]
```

In [ ]:

```
#Nan Values filled with mean of column

globalTemp.iloc[10]
```

In [ ]:

```
globalTemp.iloc[31]
```

In [ ]:

```

```