Lab 5 - Pandas

Content

- What is Pandas?
- Why Use Pandas?
- Pandas' Advantages
- Import Pandas Module
- Data Structures for Manipulating Data
- Application
- Data Cleaning
  - Empty cells
  - Data in wrong format
  - Wrong data
  - Duplicates

What is Pandas?

- Pandas is a Python library used for working with data sets.

- It has functions for analyzing, cleaning, exploring, and manipulating data.

- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

- Pandas allows us to analyze big data and make conclusions based on statistical theories.

- Pandas can clean messy data sets and make them readable and relevant.

- Relevant data is very important in data science.

Pandas' Advantages

- Fast and efficient for manipulating and analyzing data.
- Data from different file objects can be loaded.
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects

Pandas' Advantages Con…

- Data set merging and joining.
- Flexible reshaping and pivoting of data sets
- Provides time-series functionality.
- Powerful group by functionality for performing split-apply-combine operations on data sets.

Import Pandas Module

- You can import pandas into your application using the following line code.

```
import pandas as pd
```

- Here, pd is referred to as an alias to the Pandas.
- It is not necessary to import the library using alias, it just helps in writing less amount of code every time a method or property is called.

Data Structures for Manipulating Data

- Pandas generally provide two data structure for manipulating data:
  - **Series:** it is like a column in a table. It is a one-dimensional array holding data of any type.
  - **DataFrame:** it is a 2-dimensional data structure, like a 2-dimensional array, or a table with rows and columns.

Data Structures for Manipulating Data Con…

- Series Example:

```
import pandas as pd

s = pd.Series([1,2,3,4])
print(s)
print(s[2])
```

```
0    1
1    2
2    3
3    4
dtype: int64
```

```
3
```

- Create Label:

```
import pandas as pd
import numpy as np

x = np.array([15.2, 19.0, 40.6])
s2 = pd.Series(x, index=['a','b','c'])
print(s2)
print(s2['b'])
```

```
a    15.2
b    19.0
c    40.6
dtype: float64
```

```
19.0
```

Data Structures for Manipulating Data Con…

- Key/Value Objects as Series:

```
Series 1:
 day1      Sun
day2       Mon
day3      Tues
dtype: object
Series 2:
 day1      Sun
day3      Tues
dtype: object
```

```
import pandas as pd

workingDays = {"day1": 'Sun', "day2": 'Mon', "day3": 'Tues'}

s1 = pd.Series(workingDays)

print('Series 1:\n',s1)

s2 = pd.Series(workingDays, index=['day1','day3'])

print('Series 2:\n',s2)
```

Data Structures for Manipulating Data Con…

- DataFrame Example:

```
        Name  Age
Stud1    Tom   20
Stud2   nick   21
Stud3   jack   18
Stud1     Tom
Stud2    nick
Stud3    jack
Name: Name, dtype: object
```

```
import pandas as pd

# intialise data of lists.
data = {'Name':['Tom', 'nick',  'jack'],
        'Age':[20, 21, 18]}

# Create DataFrame
df = pd.DataFrame(data, index = ["Stud1", "Stud2", "Stud3"])

# Print the table.
print(df)
# Print the Name column
print(df['Name'])
```

Data Structures for Manipulating Data Con…

- Pandas Series and DataFrame can be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file.

Application

- We will work on data in this CSV file.
- Note:
  - The uploaded ".ipynb" file is opened using Colabs.

Country	Region	Requester	Date of Purchase	Total	Quantity
India	North	John	9/16/2016 0:00	100000	567
US	North	Bill	10/19/2018 0:00	120000	3000
UK	North	Thomas	6/10/2014 0:00	140000	345
Australia	East	John	11/23/2010 0:00	160000	1000
Africa	East	Bill	2/17/2010 0:00	180000	123
Singapore	East	Thomas	8/14/2017 0:00	200000	1000
Mylasia	West	John	8/3/2018 0:00	1000000	7890
India	West	Bill	7/24/2013 0:00	240000	200
US	West	Thomas	6/21/2014 0:00	26000000	1000
UK	North	John	6/26/2015 0:00	100000	1000
Australia	North	Bill	6/18/2013 0:00	120000	567
Africa	North	Thomas	9/13/2016 0:00	140000	1000
Singapore	East	John	2/14/2011 0:00	160000	892
Mylasia	East	Bill	10/5/2010 0:00	180000	444
India	East	Thomas	12/5/2012 0:00	200000	90
US	West	John	1/9/2013 0:00	220000	90
UK	West	Bill	3/4/2011 0:00	240000	90
Australia	West	Thomas	8/18/2015 0:00	260000	90
Africa	North	John	9/10/2013 0:00	140000	85
Singapore	North	Bill	3/2/2018 0:00	150000	85

Application Con…

- Reading CSV

```
df = pd.read_csv("pandas_blog.csv")
```

- A gist of the Data

```
df.head()      # Print the first 5 rows
```

	Country	Region	Requester	Date of Purchase	Total	Quantity
0	India	North	John	9/16/2016 0:00:00	100000	567
1	US	North	Bill	10/19/2018 0:00:00	120000	3000
2	UK	North	Thomas	6/10/2014 0:00:00	140000	345
3	Australia	East	John	11/23/2010 0:00:00	160000	1000
4	Africa	East	Bill	2/17/2010 0:00:00	180000	123

Application Con…

- Know your columns

```
df.columns.values    # view columns' names
```

```
array(['Country', 'Region', 'Requester', 'Date of Purchase', 'Total',
       'Quantity'], dtype=object)
```

```
df.describe()    # view some basic statistical details
```

	Total	Quantity
count	2.000000e+01	20.000000
mean	1.502500e+06	977.900000
std	5.769280e+06	1761.923497
min	1.000000e+05	85.000000
25%	1.400000e+05	90.000000
50%	1.700000e+05	505.500000
75%	2.250000e+05	1000.000000
max	2.600000e+07	7890.000000

Application Con…

- Extract a single column

```
df[["Total"]]
```

	Total
0	100000
1	120000
.	
.	
.	

(Will print the whole column values)

- Extract multiple columns

```
# df["Total", "Quantity", "Country"] # This will throw an error
df[["Total", "Quantity", "Country"]] # This will not
```

	Total	Quantity	Country
0	100000	567	India
1	120000	3000	US
2	140000	345	UK
3	160000	1000	Australia
4	180000	123	Africa
5	200000	1000	Singapore
6	1000000	7890	Mylasia
7	240000	200	India
8	26000000	1000	US
9	100000	1000	UK
10	120000	567	Australia
11	140000	1000	Africa
12	160000	892	Singapore
13	180000	444	Mylasia
14	200000	90	India
15	220000	90	US
16	240000	90	UK
17	260000	90	Australia
18	140000	85	Africa
19	150000	85	Singapore

Application Con…

- Extract multiple rows:

```
df.loc[[0, 1, 4, 5]]
```

	Country	Region	Requester	Date of Purchase	Total	Quantity
0	India	North	John	2016-09-16	100000	567
1	US	North	Bill	2018-10-19	120000	3000
4	Africa	East	Bill	2010-02-17	180000	123
5	Singapore	East	Thomas	2017-08-14	200000	1000

Application Con…

- Extract single row

```
df.iloc[0:1, :]    # Selection by passing integer location
```

	Country	Region	Requester	Date of Purchase	Total	Quantity
0	India	North	John	9/16/2016 0:00:00	100000	567

- Extract more than one row

```
df.iloc[0:3, :]    #  : means all the data
```

	Country	Region	Requester	Date of Purchase	Total	Quantity
0	India	North	John	9/16/2016 0:00:00	100000	567
1	US	North	Bill	10/19/2018 0:00:00	120000	3000
2	UK	North	Thomas	6/10/2014 0:00:00	140000	345

Application Con…

- Filtering DataFrame

```
df[df["Total"] > 200000]    # Get all rows where total >200000
```

	Country	Region	Requester	Date of Purchase	Total	Quantity
6	Mylasia	West	John	8/3/2018 0:00:00	1000000	7890
7	India	West	Bill	7/24/2013 0:00:00	240000	200
8	US	West	Thomas	6/21/2014 0:00:00	26000000	1000
15	US	West	John	1/9/2013 0:00:00	220000	90
16	UK	West	Bill	3/4/2011 0:00:00	240000	90
17	Australia	West	Thomas	8/18/2015 0:00:00	260000	90

Application Con…

- Filtering DataFrame (con…)

```
df[(df["Total"] > 200000) & (df["Country"] == "UK")]
```

	Country	Region	Requester	Date of Purchase	Total	Quantity
16	UK	West	Bill	3/4/2011 0:00:00	240000	90

```
df[(df["Total"] > 200000) & (df["Country"] == "UK")][["Country", "Region", "Total"]]
```

	Country	Region	Total
16	UK	West	240000

 # Get (Country, Region and Total)
 # where total >200000 and country == UK

Application Con….

- Statistics

```
df["Total"].sum()
```

```
30050000
```

```
df[["Total", "Quantity"]].mean()
```

```
Total         1502500.0
Quantity          977.9
dtype: float64
```

```
df[["Total", "Quantity"]].min()
```

```
Total         100000
Quantity          85
dtype: int64
```

Application Con…

- Statistics (con…)

```
df[["Total", "Quantity"]].max()
```

```
Total        26000000
Quantity         7890
dtype: int64
```

```
df[["Total", "Quantity"]].median()
```

```
Total        170000.0
Quantity        505.5
dtype: float64
```

```
df[["Total", "Quantity"]].mode()
```

	Total	Quantity
0	140000	1000

Application Con…

- Groupby country

```
df.groupby("Country").sum()
```

Country	Total	Quantity
Africa	460000	1208
Australia	540000	1657
India	540000	857
Mylasia	1180000	8334
Singapore	510000	1977
UK	480000	1435
US	26340000	4090

Application Con…

- Groupby country and region

```
df.groupby(["Country", "Region"]).sum()
```

Country	Region	Total	Quantity
Africa	East	180000	123
	North	280000	1085
Australia	East	160000	1000
	North	120000	567
	West	260000	90
India	East	200000	90
	North	100000	567
	West	240000	200
Mylasia	East	180000	444
	West	1000000	7890
Singapore	East	360000	1892
	North	150000	85
UK	North	240000	1345
	West	240000	90
US	North	120000	3000
	West	26220000	1090

Application Con…

- Just a quantity

```
df.groupby(["Country", "Region"])[["Quantity"]].sum()
```

 # Group the data by (Country, Region)
 # then get the sum of quantity values for every group

Country	Region	Quantity
Africa	East	123
	North	1085
Australia	East	1000
	North	567
	West	90
India	East	90
	North	567
	West	200
Mylasia	East	444
	West	7890
Singapore	East	1892
	North	85
UK	North	1345
	West	90
US	North	3000
	West	1090

Application Con…

- Aggregation functions

 # agg() function allows multiple statistics to be calculated per group in one calculation

```
df.groupby(["Country", "Region"]).agg({'Total':['sum', 'max'], 'Quantity':'mean'})
```

		Total		Quantity
		sum	max	mean
Country	Region			
Africa	East	180000	180000	123.0
	North	280000	140000	542.5
Australia	East	160000	160000	1000.0
	North	120000	120000	567.0
	West	260000	260000	90.0
India	East	200000	200000	90.0
	North	100000	100000	567.0
	West	240000	240000	200.0
Mylasia	East	180000	180000	444.0
	West	1000000	1000000	7890.0
Singapore	East	360000	200000	946.0
	North	150000	150000	85.0
UK	North	240000	140000	672.5
	West	240000	240000	90.0
US	North	120000	120000	3000.0
	West	26220000	26000000	545.0

Application Con…

The pivot table takes simple column-wise data as input, and groups the entries into a two-dimensional table that provides a multidimensional summarization of the data.

- Pivot tables

```
import numpy as np
df.pivot_table(index=["Country"], columns=["Region"], values=["Quantity"], aggfunc=[np.sum])
```

	sum		
	Quantity		
Region	East	North	West
Country			
Africa	123.0	1085.0	NaN
Australia	1000.0	567.0	90.0
India	90.0	567.0	200.0
Mylasia	444.0	NaN	7890.0
Singapore	1892.0	85.0	NaN
UK	NaN	1345.0	90.0
US	NaN	3000.0	1090.0

Application Con…

- Pivot tables (con…)

```
import numpy as np
df.pivot_table(index=["Country"], columns=["Region","Requester"], values=["Quantity"], aggfunc=[np.sum],
               margins=True,
              margins_name="Grand Total")
```

	sum									
	Quantity									
Region	East			North			West			Grand Total
Requester	Bill	John	Thomas	Bill	John	Thomas	Bill	John	Thomas	
Country										
Africa	123.0	NaN	NaN	NaN	85.0	1000.0	NaN	NaN	NaN	1208
Australia	NaN	1000.0	NaN	567.0	NaN	NaN	NaN	NaN	90.0	1657
India	NaN	NaN	90.0	NaN	567.0	NaN	200.0	NaN	NaN	857
Mylasia	444.0	NaN	NaN	NaN	NaN	NaN	NaN	7890.0	NaN	8334
Singapore	NaN	892.0	1000.0	85.0	NaN	NaN	NaN	NaN	NaN	1977
UK	NaN	NaN	NaN	NaN	1000.0	345.0	90.0	NaN	NaN	1435
US	NaN	NaN	NaN	3000.0	NaN	NaN	NaN	90.0	1000.0	4090
Grand Total	567.0	1892.0	1090.0	3652.0	1652.0	1345.0	290.0	7980.0	1090.0	19558

Application Con…

- Access and get the data type:

```
type(df['Date of Purchase'].iloc[0])
```

```
str
```

```
df['Date of Purchase'] = pd.to_datetime(df['Date of Purchase'])
```

```
type(df['Date of Purchase'].iloc[0])
```

```
pandas._libs.tslibs.timestamps.Timestamp
```

Application Con…

- Access datetime data

```
df['Date of Purchase'].dt.year
```

```
0     2016
1     2018
2     2014
3     2010
4     2010
5     2017
6     2018
7     2013
8     2014
9     2015
10    2013
11    2016
12    2011
13    2010
14    2012
15    2013
16    2011
17    2015
18    2013
19    2018
Name: Date of Purchase, dtype: int64
```

```
df['Date of Purchase'].dt.day
```

```
0     16
1     19
2     10
3     23
4     17
5     14
6      3
7     24
8     21
9     26
10    18
11    13
12    14
13     5
14     5
15     9
16     4
17    18
18    10
19     2
Name: Date of Purchase, dtype: int64
```

Data cleaning (Extra Info)

- Data cleaning means fixing bad data in your data set.
- Bad data could be:
  - Empty cells
  - Data in wrong format
  - Wrong data
  - Duplicates

Data Cleaning - Empty Cell

- Empty Cell can potentially give you a wrong result when you analyze data.
- Ways to deal with empty cells
  - Remove rows that contain empty cells.
  - Replace empty values

Data Cleaning - Empty Cell Con…

- Remove rows that contain empty cells.

```
newdf = df.dropna()
```

  - If you want to change the original DataFrame, use the inplace = True argument

```
df.dropna(inplace = True)
```

Data Cleaning - Empty Cell Con…

- Replace empty values

```
df.fillna(130, inplace = True)
```

  - Replace in specific column

```
df["Quantity"].fillna(130, inplace = True)
```

  - Replace Using Mean, Median, or Mode:

    - A common way to replace empty cells, is to calculate the mean, median or mode value of the column.

```
x = df["Quantity"].mean()

df["Quantity"].fillna(x, inplace = True)
```

Data Cleaning – Wrong Format

- To fix wrong format data, you have two options:
  - Remove the rows

```
df.dropna(subset=['Date of Purchase'], inplace = True)
#Null value in date time = NaT
```

  - Convert all cells in the columns into the same format

```
df['Date of Purchase'] = pd.to_datetime(df['Date of Purchase'])
```

Data Cleaning – Wrong Data

- Example: If you have a data set for courses in the college. You have class duration is 2 or 3 hours. While you check the data set you find out that there is a classes have duration 30 hours.
- We could conclude that it is impossible to have a class duration for 30 hours. So, you need to fix this wrong data.

Data Cleaning – Wrong Data Con…

- Ways to deal with wrong data
  - Replace wrong data

```
# Replace Duration value in row x
for x in df.index:
  if df.loc[x, "Class Duration"] > 2:
    df.loc[x, "Class Duration"] = 2
```

  - Remove rows

```
for x in df.index:
  if df.loc[x, "Class Duration"] > 2:
    df.drop(x, inplace = True)
```

Data Cleaning – Duplicates

- Duplicate rows are rows that have been registered more than one time.
- Discovering duplicates:

```
df.duplicated()
```

- Remove duplicates

```
df.drop_duplicates(inplace = True)
```

Thanks