CPE 352 Data Science

3 – Data Preparation

Asst. Prof. Dr. Santitham Prom-on

Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi

Lecture 3 – Overview

- Pandas for data preparation
- Data wrangling practice – building an analytic-based table (RFM)

Google Colab

https://drive.google.com/file/d/1X-mW6y08WxiXioggH2Tplw94HbLUizJA/view?usp=sharing

BIG DATA
EXPERIENCE

Pandas

Part I

KMUTT

Data Wrangling =
Data Preparation + Exploratory Data Analysis

- Data (especially transactional data) are needed to be prepared into specific format for analysis
- While preparing, exploratory data analysis or “EDA” is a critical step in analyzing the data
- The main reasons are
  - detection of mistakes, outliers or abnormalities
  - checking of assumptions
  - preliminary selection of appropriate models
  - determining relationships among the explanatory variables
  - assessing the direction and rough size of relationship explanatory and outcome variables

BIG DATA
EXPERIENCE

pandas: Python Data Analysis

- Pandas is one of the most widely used Python libraries in data science.

- The name, pandas, comes from ***pan****el* ***da****ta*, a common term for multidimensional data sets encountered in statistics and econometrics.

KMUTT

Pandas DataFrame

- In pandas, data frame is a way to store data in rectangular grids that can easily be overviewed and processed.
  - Each row of these grids corresponds to measurements or values of an instance, while each column is a vector containing data for a specific variable.
  - This means that a data frame's rows do not need to contain the same type of values: they can be numeric, character, logical, etc.

Pandas data frame

- Pandas data frame consists of three main components: the data, the index, and the columns.
- The DataFrame can contain data that is:
  - a Pandas DataFrame
  - a Pandas Series: a one-dimensional labeled array capable of holding any data type with axis labels or index. An example of a Series object is one column from a DataFrame.
  - a Numpy ndarray, which can be a record or structured
  - a two-dimensional ndarray
  - dictionaries of one-dimensional ndarrays, lists, dictionaries or Series.

BIG DATA EXPERIENCE

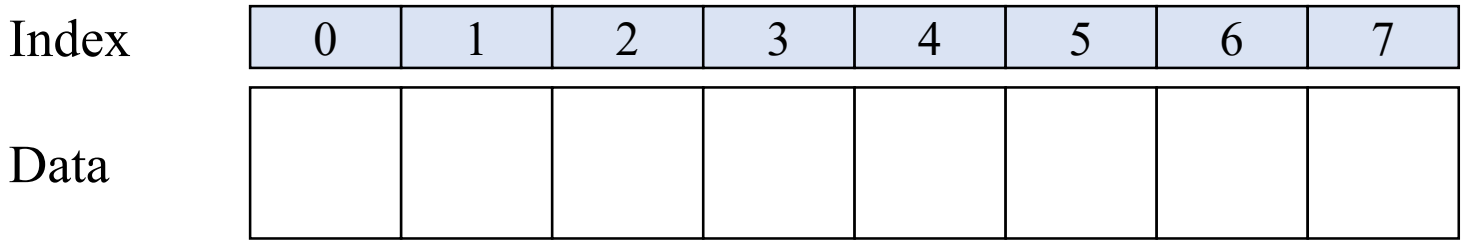
Index and Column Names

- Besides the data that your DataFrame needs to contain, you can also specify the index and column names.
- The index indicates the difference in rows, while the column names indicate the difference in columns.
- We will see later that these two components of the DataFrame are handy when you're manipulating your data.

KMUTT

Pandas Data Structure
Series

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



Example

```
s = pd/Series)np/random/randn)6*-" index>B(a(-" (b(-" (c(-" (d(-" (e(D*
```

Pandas Data Structure DataFrame

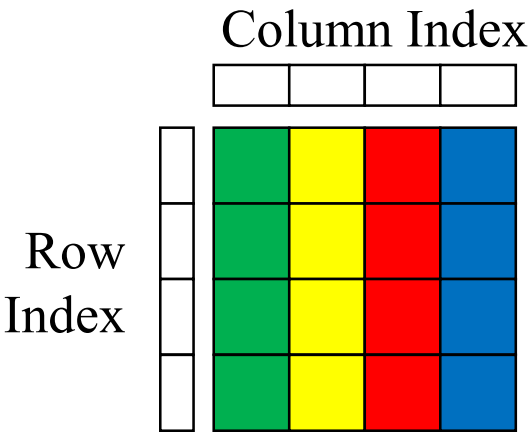
- **DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects.

Example

```
import pandas as pd

d = {'one' : [1., 2., 3., 4.],
     'two' : [4., 3., 2., 1.]}
df = pd.DataFrame(d)
print(df)
```

```
   one  two
0  1.0  4.0
1  2.0  3.0
2  3.0  2.0
3  4.0  1.0
```



Creating DataFrame from NumPy array

- You can just pass NumPy ndarray to the DataFrame() function in the data argument.

```
data = np.array([['','Col1','Col2'],
                 ['Row1',1,2],
                 ['Row2',3,4]])
df = pd.DataFrame(data=data[1:,1:],
                  index=data[1:,0],
                  columns=data[0,1:])
print(df)
```

```
      Col1 Col2
Row1     1    2
Row2     3    4
```

BIG DATA EXPERIENCE

Creating DataFrame from files

```
df = pd.read_excel('Superstore.xlsx',
                   sheet_name='Order',
                   index_col='Row ID')
df
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region
1	CA-2013-152156	2014-11-09	2014-11-12	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420	South
2	CA-2013-152156	2014-11-09	2014-11-12	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420	South
3	CA-2013-138688	2014-06-13	2014-06-17	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036	West
4	US-2012-108966	2013-10-11	2013-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311	South

KMUTT

DataFrame dimension

- After you have created your DataFrame, you might want to know a little bit more about it.
- You can use the shape property .shape:

```
df.shape
```

```
(9994, 20)
```

BIG DATA EXPERIENCE

Viewing DataFrame: head()

```
df.head()
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code
1	CA-2013-152156	2014-11-09	2014-11-12	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420
2	CA-2013-152156	2014-11-09	2014-11-12	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420

KMUTT

BIG DATA EXPERIENCE

Viewing DataFrame: tail()

```
df.tail()
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code
9990	CA-2011-110422	2012-01-22	2012-01-24	Second Class	TB-21400	Tom Boeckenhauer	Consumer	United States	Miami	Florida	33180
9991	CA-2014-121258	2015-02-27	2015-03-04	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa Mesa	California	92627
9992	CA-2014-121258	2015-02-27	2015-03-04	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa Mesa	California	92627
9993	CA-2014-121258	2015-02-27	2015-03-04	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa Mesa	California	92627

KMUTT

BIG DATA EXPERIENCE

Viewing DataFrame: columns

```
df.columns
```

```
Index(['Order ID', 'Order Date', 'Ship Date', 'Ship Mode', 'Customer ID',
       'Customer Name', 'Segment', 'Country', 'City', 'State', 'Postal Code',
       'Region', 'Product ID', 'Category', 'Sub-Category', 'Product Name',
       'Sales', 'Quantity', 'Discount', 'Profit'],
      dtype='object')
```

KMUTT

Viewing DataFrame: describe()

```
df.describe()
```

	Postal Code	Sales	Quantity	Discount	Profit
count	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000
mean	55190.379428	229.858001	3.789574	0.156203	28.656896
std	32063.693350	623.245101	2.225110	0.206452	234.260108
min	1040.000000	0.444000	1.000000	0.000000	-6599.978000
25%	23223.000000	17.280000	2.000000	0.000000	1.728750
50%	56430.500000	54.490000	3.000000	0.200000	8.666500
75%	90008.000000	209.940000	5.000000	0.200000	29.364000
max	99301.000000	22638.480000	14.000000	0.800000	8399.976000

Sorting data by specific columns

```
df.sort_values(by=['Customer ID','Ship Date'])
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City
2230	CA-2011-128055	2012-03-31	2012-04-05	Standard Class	AA-10315	Alex Avila	Consumer	United States	San Francisco
2231	CA-2011-128055	2012-03-31	2012-04-05	Standard Class	AA-10315	Alex Avila	Consumer	United States	San Francisco
7469	CA-2011-138100	2012-09-15	2012-09-20	Standard Class	AA-10315	Alex Avila	Consumer	United States	New York City

Select a column

- You can simply select the column using the column name

```
df['Segment']
```

```
Row ID
1            Consumer
2            Consumer
3           Corporate
4            Consumer
5            Consumer
6            Consumer
7            Consumer
8            Consumer
```

```
df.Segment
```

```
Row ID
1            Consumer
2            Consumer
3           Corporate
4            Consumer
5            Consumer
6            Consumer
7            Consumer
8            Consumer
```

Select multiple columns

- Multiple columns can be selected using a list

```
df[['Order Date','Sales']]
```

Row ID	Order Date	Sales
1	2014-11-09	261.9600
2	2014-11-09	731.9400
3	2014-06-13	14.6200
4	2013-10-11	957.5775
5	2013-10-11	22.3680

More complex selections .loc and iloc

- You can use .loc and .iloc to perform complex selection by labels and index respectively

```
df.loc[[2,4,5],['Order Date','Sales']]
```

Row ID	Order Date	Sales
2	2014-11-09	731.9400
4	2013-10-11	957.5775
5	2013-10-11	22.3680

```
df.iloc[[2,4,5],[0,1,2]]
```

Row ID	Order ID	Order Date	Ship Date
3	CA-2013-138688	2014-06-13	2014-06-17
5	US-2012-108966	2013-10-11	2013-10-18
6	CA-2011-115812	2012-06-09	2012-06-14

Indexing (aka filter)

- Simple indexing can be done directly or using .loc or .iloc

```
df[0:3]
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State
1	CA-2013-152156	2014-11-09	2014-11-12	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky
2	CA-2013-152156	2014-11-09	2014-11-12	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky
3	CA-2013-138688	2014-06-13	2014-06-17	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California

Boolean indexing by isin

```
df[df['Ship Mode'].isin(['First Class'])]
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State
36	CA-2013-117590	2014-12-09	2014-12-11	First Class	GH-14485	Gene Hale	Corporate	United States	Richardson	Texas
37	CA-2013-117590	2014-12-09	2014-12-11	First Class	GH-14485	Gene Hale	Corporate	United States	Richardson	Texas
45	CA-2013-118255	2014-03-12	2014-03-14	First Class	ON-18715	Odella Nelson	Corporate	United States	Eagan	Minnesota

Boolean indexing by condition

```
df[df['Profit'] < 0]
```

	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales	Quantity	Discount	Profit
	Fort uderdale	Florida	33311	South	FUR-TA-10000577	Furniture	Tables	Bretford CR4500 Series Slim Rectangular Table	957.5775	5	0.45	-383.0310
	ort Worth	Texas	76106	Central	OFF-AP-10002311	Office Supplies	Appliances	Holmes Replacement Filter for HEPA Air Cleaner...	68.8100	5	0.80	-123.8580

Add column

```
df['Unit Sales'] = df['Sales']/df['Quantity']
df[['Product ID','Unit Sales']]
```

Row ID	Product ID	Unit Sales
1	FUR-BO-10001798	130.9800
2	FUR-CH-10000454	243.9800
3	OFF-LA-10000240	7.3100
4	FUR-TA-10000577	191.5155
5	OFF-ST-10000760	11.1840
6	FUR-FU-10001487	6.9800
7	OFF-AR-10002833	1.8200
8	TEC-PH-10002275	151.1920
9	OFF-BI-10003910	6.1680

BIG DATA EXPERIENCE

Delete column (safely)

```
df1 = df.drop(columns=['Order Date','Ship Date'])
df1.columns
```

```
Index(['Order ID', 'Ship Mode', 'Customer ID', 'Customer Name', 'Segment',
       'Country', 'City', 'State', 'Postal Code', 'Region', 'Product ID',
       'Category', 'Sub-Category', 'Product Name', 'Sales', 'Quantity',
       'Discount', 'Profit', 'Unit Sales'],
      dtype='object')
```

KMUTT

Missing data

- pandas primarily uses the value np.nan to represent missing data. It is by default not included in computations.

```
churnData = pd.read_csv('Telco-Churn.csv',na_values=' ')
churnData[churnData['tenure']==0]
```

TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
Yes	Yes	No	Two year	Yes	Bank transfer (automatic)	52.55	NaN	No
No internet service	No internet service	No internet service	Two year	No	Mailed check	20.25	NaN	No
No	Yes	Yes	Two year	No	Mailed check	80.85	NaN	No
No internet service	No internet service	No internet service	Two year	No	Mailed check	25.75	NaN	No
Yes	Yes	No	Two year	No	Credit card (automatic)	56.05	NaN	No
No internet service	No internet service	No internet service	Two year	No	Mailed check	19.85	NaN	No

Remove missing data rows

```
churnData.shape
```

```
(7043, 21)
```

```
churnData_1 = churnData.dropna(how='any')
churnData_1.shape
```

```
(7032, 21)
```

```
churnData_1.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	

Fill missing data: investigate

```
churnData.isnull().any()
```

```
customerID          False
gender              False
SeniorCitizen       False
Partner             False
Dependents          False
tenure              False
PhoneService        False
MultipleLines       False
InternetService     False
OnlineSecurity      False
OnlineBackup        False
DeviceProtection    False
TechSupport         False
StreamingTV         False
StreamingMovies     False
Contract            False
PaperlessBilling    False
PaymentMethod       False
MonthlyCharges      False
TotalCharges         True
Churn               False
dtype: bool
```

```
churnData[churnData['TotalCharges'].isnull()]
```

TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling
Yes	Yes	No	Two year	Yes
No internet service	No internet service	No internet service	Two year	No
No	Yes	Yes	Two year	No
No internet service	No internet service	No internet service	Two year	No
Yes	Yes	No	Two year	No
No internet service	No internet service	No internet service	Two year	No
No internet service	No internet service	No internet service	Two year	No

Fill missing data: fillna

```
flightData_2 = churnData.fillna(value={'TotalCharges':0})
flightData_2.isnull().any()
```

```
customerID          False
gender              False
SeniorCitizen       False
Partner             False
Dependents          False
tenure              False
PhoneService        False
MultipleLines       False
InternetService     False
OnlineSecurity      False
OnlineBackup        False
DeviceProtection    False
TechSupport         False
StreamingTV         False
StreamingMovies     False
Contract            False
PaperlessBilling    False
PaymentMethod       False
MonthlyCharges      False
TotalCharges        False
Churn               False
dtype: bool
```

Statistical Operations

```
churnData.mean()
```

```
SeniorCitizen         0.162147
tenure               32.371149
MonthlyCharges       64.761692
TotalCharges       2283.300441
dtype: float64
```

```
churnData.std()
```

```
SeniorCitizen         0.368612
tenure               24.559481
MonthlyCharges       30.090047
TotalCharges       2266.771362
dtype: float64
```

```
churnData.median()
```

```
SeniorCitizen         0.000
tenure               29.000
MonthlyCharges       70.350
TotalCharges       1397.475
dtype: float64
```

```
churnData.quantile([0.1,0.9])
```

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
0.1	0.0	2.0	20.05	84.60
0.9	1.0	69.0	102.60	5976.64

BIG DATA EXPERIENCE

Apply

```
churnData_num = churnData._get_numeric_data()
churnData_num.apply(lambda x:x/x.max())
```

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
0	0.0	0.013889	0.251368	0.003437
1	0.0	0.472222	0.479579	0.217564
2	0.0	0.027778	0.453474	0.012453
3	0.0	0.625000	0.356211	0.211951
4	0.0	0.027778	0.595368	0.017462
5	0.0	0.111111	0.839158	0.094475
6	0.0	0.305556	0.750316	0.224461
7	0.0	0.138889	0.250526	0.034762

KMUTT

BIG DATA EXPERIENCE

Transform

```
churnData.transform({'TotalCharges':np.log,
                     'PaperlessBilling':lambda x: 1 if x=='Yes' else 0})
```

	TotalCharges	PaperlessBilling
0	3.396185	1
1	7.544068	0
2	4.683519	1
3	7.517928	0
4	5.021575	1
5	6.709914	1
6	7.575277	1
7	5.710096	0

KMUTT

Concatenate

```
pd.concat([churnData['customerID'],
           churnData[['TotalCharges', 'MonthlyCharges']]],
          axis = 1)
```

	customerID	TotalCharges	MonthlyCharges
0	7590-VHVEG	29.85	29.85
1	5575-GNVDE	1889.50	56.95
2	3668-QPYBK	108.15	53.85
3	7795-CFOCW	1840.75	42.30
4	9237-HQITU	151.65	70.70
5	9305-CDSKC	820.50	99.65
6	1452-KIOVK	1949.40	89.10
7	6713-OKOMC	301.90	29.75

BIG DATA EXPERIENCE

Join

```
flights = pd.read_csv('Flight_flights.csv', index_col=0)
airlines = pd.read_csv('Flight_airlines.csv', index_col = 0)
```

```
flightData = pd.merge(flights[['time_hour','carrier','flight']],
                      airlines, on='carrier', how='left')
```

	time_hour	carrier	flight	name
0	2013-01-01 05:00:00	UA	1545	United Air Lines Inc.
1	2013-01-01 05:00:00	UA	1714	United Air Lines Inc.
2	2013-01-01 05:00:00	AA	1141	American Airlines Inc.
3	2013-01-01 05:00:00	B6	725	JetBlue Airways
4	2013-01-01 06:00:00	DL	461	Delta Air Lines Inc.
5	2013-01-01 05:00:00	UA	1696	United Air Lines Inc.
6	2013-01-01 06:00:00	B6	507	JetBlue Airways
7	2013-01-01 06:00:00	EV	5708	ExpressJet Airlines Inc.

KMUTT

Grouping

- By “group by” we are referring to a process involving one or more of the following steps
  - **Splitting** the data into groups based on some criteria
  - **Applying** a function to each group independently
  - **Combining** the results into a data structure

BIG DATA EXPERIENCE

groupby + agg

```
flights.groupby('carrier').agg({'dep_delay':[np.mean,np.max],
                                'arr_delay':[np.mean,np.max]})
```

	dep_delay		arr_delay	
	mean	amax	mean	amax
carrier				
9E	16.725769	747.0	7.379669	744.0
AA	8.586016	1014.0	0.364291	1007.0
AS	5.804775	225.0	-9.930889	198.0
B6	13.022522	502.0	9.457973	497.0
DL	9.264505	960.0	1.644341	931.0
EV	19.955390	548.0	15.796431	577.0

KMUTT

Reshape: Pivot
Transforming long- to wide-format

```
flights.groupby(['carrier','month'])\
       .dep_delay.mean()\
       .reset_index()\
       .pivot(index='carrier',columns='month',values='dep_delay')
```

month	1	2	3	4	5	6	7	8	9	10
carrier										
9E	16.882510	16.486327	13.407530	13.567164	22.672190	28.952978	31.398827	17.296807	7.754232	9.334348
AA	6.932358	8.276923	8.700291	11.696207	9.664621	14.627778	12.112621	7.169965	5.694272	3.002217
AS	7.354839	0.722222	8.419355	11.316667	6.774194	13.083333	2.419355	2.870968	-4.516667	0.677419
B6	9.493436	13.772911	14.240690	15.165175	9.778560	20.392170	24.902315	15.678593	6.634260	2.963065
DL	3.849768	5.537440	9.933430	8.166544	9.741168	18.735941	20.582242	9.846974	5.526071	3.417502
EV	24.228879	21.523328	26.169820	22.767549	20.242477	25.496834	26.504722	16.261828	8.237970	13.418050
F9	10.000000	29.770833	16.754386	24.631579	35.948276	29.436364	31.810345	22.218182	8.263158	9.701754
FL	1.972222	5.180851	17.252459	13.121311	19.183230	38.806584	41.162698	23.410156	16.948819	13.679487
HA	54.387097	17.357143	1.161290	-2.100000	-1.451613	1.466667	-1.709677	1.677419	-5.440000	-5.095238
MQ	6.485494	8.092962	7.193262	13.738095	13.925859	20.842342	20.745310	10.050277	5.350545	4.478957
OO	67.000000	NaN	NaN	NaN	NaN	61.000000	NaN	64.000000	-4.941176	NaN

Reshaping: Melt (1)
Transforming wide- to long-format

```
expenditure = pd.read_excel('Gov_Expenditure_EDU.xls', sheet_name='Data')
```

```
expenditure.head()
```

	Country N	Country	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	1967	1968	1969	1970
0	Aruba	ABW	Government expenditure on education, total (% ...	SE.XPD.TOTL.GB.ZS	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Andorra	AND	Government expenditure on education, total (% ...	SE.XPD.TOTL.GB.ZS	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Afghanistan	AFG	Government expenditure on education, total (% ...	SE.XPD.TOTL.GB.ZS	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Reshaping: Melt (2)
Transforming wide- to long-format

```
expenditureByCountry = expenditure.drop(columns=['Indicator Name','Indicator Code'])\
                                  .melt(id_vars='Country Name',
                                        value_vars=np.arange(1960,2015,1).astype('str'),
                                        var_name='Year',
                                        value_name='Amount')\
                                  .sort_values(by=['Country Name','Year'])
expenditureByCountry[expenditureByCountry['Country Name']=='Thailand']
```

	Country Name	Year	Amount
219	Thailand	1960	NaN
468	Thailand	1961	NaN
717	Thailand	1962	NaN
966	Thailand	1963	NaN
1215	Thailand	1964	NaN
1464	Thailand	1965	NaN
1713	Thailand	1966	NaN
1962	Thailand	1967	NaN

BIG DATA EXPERIENCE

Time-Series
Set time as index

```
flightData['time_hour'] = pd.to_datetime(flightData['time_hour'])
```

```
flightData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 336776 entries, 0 to 336775
Data columns (total 4 columns):
time_hour    336776 non-null datetime64[ns]
carrier      336776 non-null object
flight       336776 non-null int64
name         336776 non-null object
dtypes: datetime64[ns](1), int64(1), object(2)
memory usage: 12.8+ MB
```

```
flightData.set_index('time_hour',inplace=True)
```

KMUTT

BIG DATA EXPERIENCE

Time Series Aggregate

```
flightData.resample('W').carrier.count()
```

```
time_hour
2013-01-06    5166
2013-01-13    6114
2013-01-20    6034
2013-01-27    6049
2013-02-03    6063
2013-02-10    6104
2013-02-17    6236
2013-02-24    6381
2013-03-03    6444
2013-03-10    6546
2013-03-17    6555
2013-03-24    6547
2013-03-31    6550
```

```
flightData.resample('M').carrier.count()
```

```
time_hour
2013-01-31    27004
2013-02-28    24951
2013-03-31    28834
2013-04-30    28330
2013-05-31    28796
2013-06-30    28243
2013-07-31    29425
2013-08-31    29327
2013-09-30    27574
2013-10-31    28889
2013-11-30    27268
2013-12-31    28135
```

KMUTT

Dummy variables

- To convert a categorical variable into a “dummy” or “indicator” DataFrame

```
pd.get_dummies(churnData['Contract'],prefix='Contract')
```

	Contract_Month-to-month	Contract_One year	Contract_Two year
0	1	0	0
1	0	1	0
2	1	0	0
3	0	1	0
4	1	0	0
5	1	0	0
6	1	0	0
7	1	0	0
8	1	0	0

Dummy variables
k-1 columns for k levels

- A kth dummy variable is redundant
- It carries no new information.
- It creates a severe multicollinearity problem.
- For k-level category, only k - 1 dummy variables are required.

```
pd.get_dummies(churnData['Contract'],
               prefix='Contract',
               drop_first=True)
```

	Contract_One year	Contract_Two year
0	0	0
1	1	0
2	0	0
3	1	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	1	0

BIG DATA EXPERIENCE

Sample data

```
churnData.sample(frac=0.1)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
4192	7137-NAXML	Male	0	No	No	23	Yes	No
675	3729-OWRVL	Male	1	No	No	65	Yes	Yes
3181	1265-ZFOSD	Female	0	Yes	No	64	Yes	Yes
3648	9419-IPPBE	Female	0	Yes	Yes	51	Yes	Yes
2102	2911-WDXMV	Male	0	No	Yes	18	Yes	Yes
860	4795-UXVCJ	Male	0	No	No	26	Yes	No
2002	0017-DINOC	Male	0	No	No	54	No	No phone service
6887	2511-ALLCS	Female	0	Yes	Yes	35	Yes	Yes

KMUTT

Sample data: stratified

```
churnData.groupby('Churn', group_keys=False).apply(lambda x: x.sample(n=2))
```

TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
Yes	Yes	Yes	One year	Yes	Bank transfer (automatic)	105.05	6004.85	No
Yes	Yes	Yes	One year	Yes	Electronic check	99.25	3532.00	No
No	Yes	No	Month-to-month	Yes	Electronic check	86.25	587.10	Yes
No	Yes	No	Month-to-month	Yes	Credit card (automatic)	83.15	446.05	Yes

Variable binning

```
pd.cut(churnData['TotalCharges'],5)
```

```
0        (10.134, 1752.0]
1        (1752.0, 3485.2]
2        (10.134, 1752.0]
3        (1752.0, 3485.2]
4        (10.134, 1752.0]
5        (10.134, 1752.0]
6        (1752.0, 3485.2]
7        (10.134, 1752.0]
8        (1752.0, 3485.2]
9        (3485.2, 5218.4]
10       (10.134, 1752.0]
11       (10.134, 1752.0]
12       (5218.4, 6951.6]
13       (3485.2, 5218.4]
14       (1752.0, 3485.2]
15       (6951.6, 8684.8]
```

Variable binning: custom bins

```
In [65]: pd.cut(churnData['TotalCharges'],bins=[0, 10, 50, 1000, 5000, 10000])

Out[65]: 0            (10, 50]
         1          (1000, 5000]
         2            (50, 1000]
         3          (1000, 5000]
         4            (50, 1000]
         5            (50, 1000]
         6          (1000, 5000]
         7            (50, 1000]
         8          (1000, 5000]
         9          (1000, 5000]
         10           (50, 1000]
         11           (50, 1000]
         12        (5000, 10000]
```

Variable binning: quantile

```
pd.qcut(churnData['TotalCharges'],q=4)
```

```
0          (18.799, 401.45]
1       (1397.475, 3794.738]
2          (18.799, 401.45]
3       (1397.475, 3794.738]
4          (18.799, 401.45]
5         (401.45, 1397.475]
6       (1397.475, 3794.738]
7          (18.799, 401.45]
8       (1397.475, 3794.738]
9       (1397.475, 3794.738]
10        (401.45, 1397.475]
11         (18.799, 401.45]
12        (3794.738, 8684.8]
13        (3794.738, 8684.8]
14      (1397.475, 3794.738]
```

Separating formatted string into multiple columns (1)

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

```
movies.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Separating formatted string into multiple columns (2)

```
genres = movies['genres'].str.split('|',expand=True)
maxcol = genres.shape[1]
movies1 = pd.concat([movies, genres] , axis=1)
movies2 = movies1.melt(id_vars=['movieId','title'],
                       value_vars=np.arange(0,maxcol,1),
                       value_name='genre')
movies2
```

	movieId	title	variable	genre
0	1	Toy Story (1995)	0	Adventure
1	2	Jumanji (1995)	0	Adventure
2	3	Grumpier Old Men (1995)	0	Comedy
3	4	Waiting to Exhale (1995)	0	Comedy
4	5	Father of the Bride Part II (1995)	0	Comedy
5	6	Heat (1995)	0	Action

Separating formatted string into multiple columns (3)

```
genres_1 = movies2.dropna()\
                  .pivot(index='movieId',columns='genre',values='present')\
                  .fillna(0)
pd.merge(movies,genres_1,on='movieId').drop(columns='(no genres listed)')
```

	movieId	title	genres	Action	Adventure	Animation	Children
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0.0	1.0	1.0	1.0
1	2	Jumanji (1995)	Adventure|Children|Fantasy	0.0	1.0	0.0	1.0
2	3	Grumpier Old Men (1995)	Comedy|Romance	0.0	0.0	0.0	0.0
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	0.0	0.0	0.0	0.0

BIG DATA
EXPERIENCE

RFM (Recency-Frequency-Monetary)

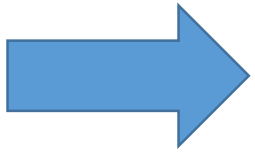
Data Wrangling Practice 1

KMUTT

BIG DATA EXPERIENCE

Goal

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment
1	CA-2013-152156	2014-11-09	2014-11-12	Second Class	CG-12520	Claire Gute	Consumer
2	CA-2013-152156	2014-11-09	2014-11-12	Second Class	CG-12520	Claire Gute	Consumer
3	CA-2013-138688	2014-06-13	2014-06-17	Second Class	DV-13045	Darrin Van Huff	Corporate
4	US-2012-108966	2013-10-11	2013-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer
5	US-2012-108966	2013-10-11	2013-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer



Customer ID	R	F	M
AA-10315	3	0	3
AA-10375	0	3	0
AA-10480	3	0	1
AA-10645	1	1	3
AB-10015	3	0	0
AB-10060	1	2	3
AB-10105	1	3	3
AB-10150	1	0	0
AB-10165	0	2	0
AB-10255	2	3	0
AB-10600	1	0	1
AC-10420	3	0	0

KMUTT

BIG DATA EXPERIENCE

1.1 Load data

```
import pandas as pd

superstore = pd.read_excel('./Data/superstore.xlsx',
                           index_col='Row ID')
superstore.head()
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
1	CA-2013-152156	2014-11-09	2014-11-12	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420	South	FUR-BO-10001798	Furniture
2	CA-2013-152156	2014-11-09	2014-11-12	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420	South	FUR-CH-10000454	Furniture
3	CA-2013-138688	2014-06-13	2014-06-17	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036	West	OFF-LA-10000240	Office Supplies

KMUTT

BIG DATA EXPERIENCE

1.2 Check data types

```
superstore.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9994 entries, 1 to 9994
Data columns (total 20 columns):
Order ID          9994 non-null object
Order Date        9994 non-null datetime64[ns]
Ship Date         9994 non-null datetime64[ns]
Ship Mode         9994 non-null object
Customer ID       9994 non-null object
Customer Name     9994 non-null object
Segment           9994 non-null object
Country           9994 non-null object
City              9994 non-null object
State             9994 non-null object
Postal Code       9994 non-null int64
Region            9994 non-null object
Product ID        9994 non-null object
Category          9994 non-null object
Sub-Category      9994 non-null object
Product Name      9994 non-null object
Sales             9994 non-null float64
Quantity          9994 non-null int64
Discount          9994 non-null float64
Profit            9994 non-null float64
dtypes: datetime64[ns](2), float64(3), int64(2), object(13)
memory usage: 1.6+ MB
```

KMUTT

1.3 Get current date

```
currentDate = superstore['Order Date'].max()
```

```
(currentDate - superstore['Order Date']).dt.days
```

```
Row ID
1          417
2          417
3          566
4          811
5          811
6         1300
7         1300
8         1300
9         1300
10        1300
11        1300
12        1300
13         259
14         390
```

BIG DATA EXPERIENCE

1.4 Define recency function

```
def recency(x):
    return (currentDate - x.max()).days
```

```
recency(superstore['Order Date'])
```

```
0
```

KMUTT

1.5 Recency of each customer

```
superstore.groupby('Customer ID').agg({'Order Date':recency})
```

Customer ID	Order Date
AA-10315	184
AA-10375	19
AA-10480	259
AA-10645	55
AB-10015	415
AB-10060	54
AB-10105	41
AB-10150	41
AB-10165	25
AB-10255	166

1.6 Add recency of each customer

```
superstore.groupby('Customer ID').agg({'Order Date':recency, 'Order ID':'nunique'})
```

Customer ID	Order Date	Order ID
AA-10315	184	5
AA-10375	19	9
AA-10480	259	4
AA-10645	55	6
AB-10015	415	3
AB-10060	54	8
AB-10105	41	10
AB-10150	41	5
AB-10165	25	8

1.7 Add monetary of each customer

```
superstore.groupby('Customer ID').agg({'Order Date':recency,
                                       'Order ID':'nunique',
                                       'Sales':sum})
```

Customer ID	Order Date	Order ID	Sales
AA-10315	184	5	5563.5600
AA-10375	19	9	1056.3900
AA-10480	259	4	1790.5120
AA-10645	55	6	5086.9350
AB-10015	415	3	886.1560
AB-10060	54	8	7755.6200
AB-10105	41	10	14473.5710
AB-10150	41	5	966.7100
AB-10165	25	8	1113.8380

1.8 Rename columns

```
superstore\
    .groupby('Customer ID')\
    .agg({'Order Date':recency,'Order ID':'nunique','Sales':sum})\
    .rename(columns={'Order Date': 'R', 'Order ID':'F', 'Sales':'M'})
```

Customer ID	R	F	M
AA-10315	184	5	5563.5600
AA-10375	19	9	1056.3900
AA-10480	259	4	1790.5120
AA-10645	55	6	5086.9350
AB-10015	415	3	886.1560
AB-10060	54	8	7755.6200
AB-10105	41	10	14473.5710
AB-10150	41	5	966.7100
AB-10165	25	8	1113.8380

1.9 Convert values to quartiles

```
rfmData = superstore\
    .groupby('Customer ID')\
    .agg({'Order Date':recency,'Order ID':'nunique','Sales':sum})\
    .rename(columns={'Order Date': 'R', 'Order ID':'F', 'Sales':'M'})
```

```
rfmData.transform(lambda x: pd.qcut(x, q=4,labels=False))
```

Customer ID	R	F	M
AA-10315	3	0	3
AA-10375	0	3	0
AA-10480	3	0	1
AA-10645	1	1	3
AB-10015	3	0	0
AB-10060	1	2	3
AB-10105	1	3	3
AB-10150	1	0	0
AB-10165	0	2	0

High value customer that has not made any purchase for a long time

Low value customer with frequency purchase (small-ticket customer)

High value, frequent buyer, and recent visit customer

BIG DATA EXPERIENCE

Lab

- Load two data frames “Flight_flights.csv” and “Flight_airlines.csv”
- Select two columns (carrier, dep_delay)
- Filter NA out of dep_delay
- Use carrier as a group and calculate mean of dep_delay
- Sort the worst carrier by departure delay (dep_delay)
- Join airlines data to flight data using ‘carrier’ as a key
- Show the result

KMUTT

BIG DATA
EXPERIENCE

Thank you

Question?

KMUTT