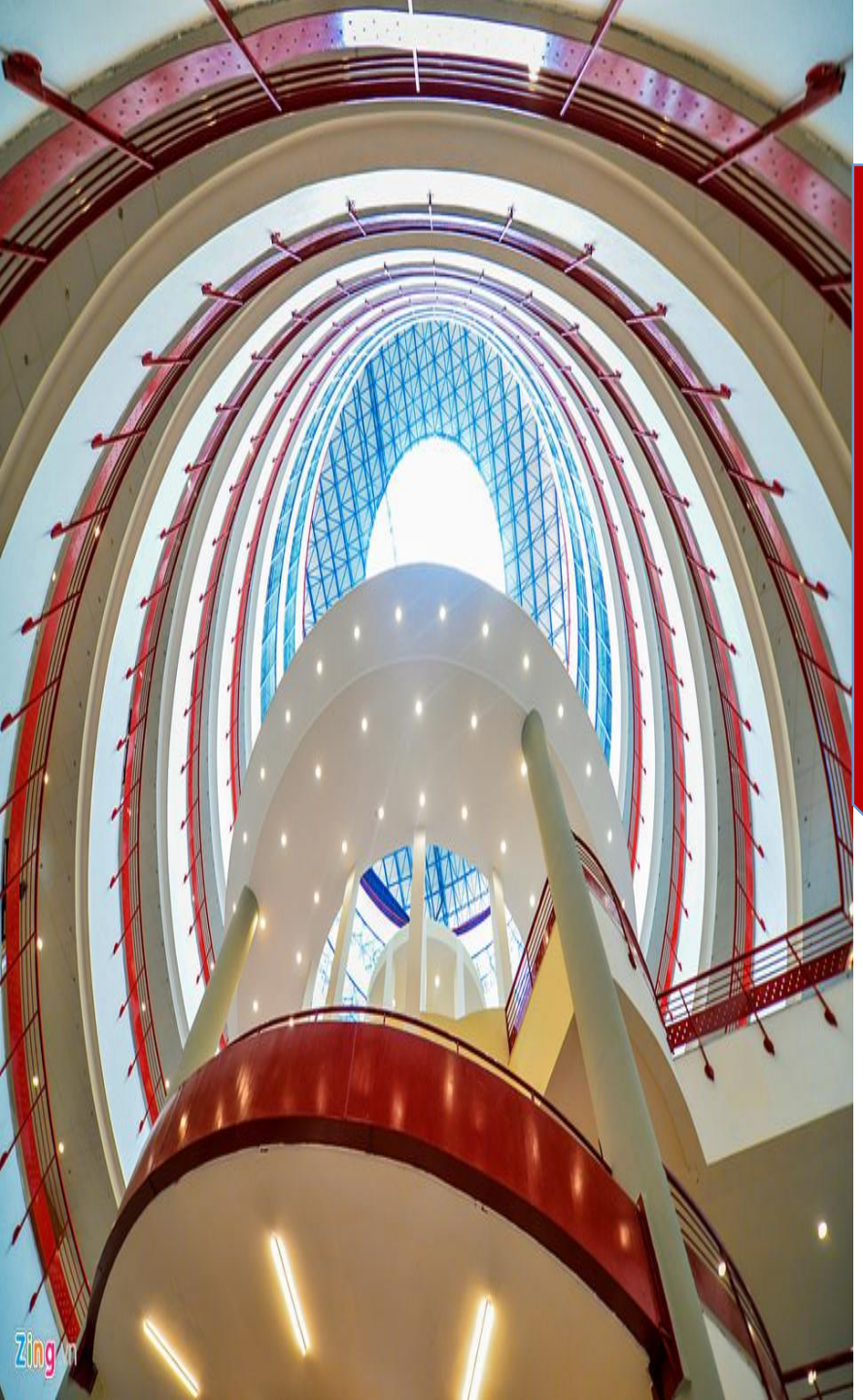
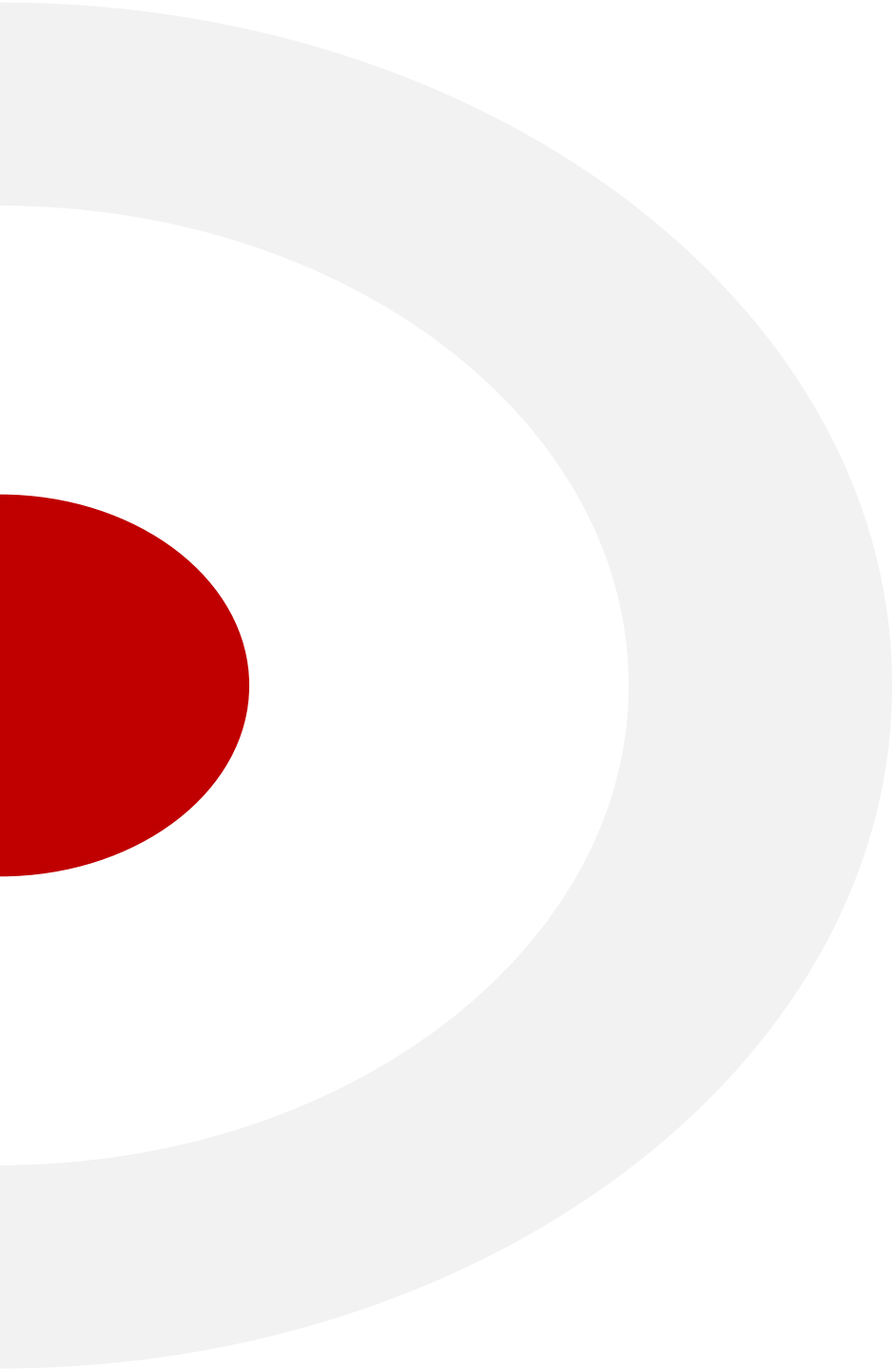


DATA PREPARATION AND VISUALIZATION

Department of Mathematical Economics

National Economics University
https://www.neu.edu.vn/



Combining Data Using Pandas

Combining and Merging Datasets

- Data contained in pandas objects can be combined together in a number of ways:
  - Pandas.merge connects rows in DataFrame based on one or more keys
  - Pandas.concat concatenates or “stacks” together objects along an axis
  - The combine_first instance method enables splicing together overlapping data to fill in missing values in one object with values from another

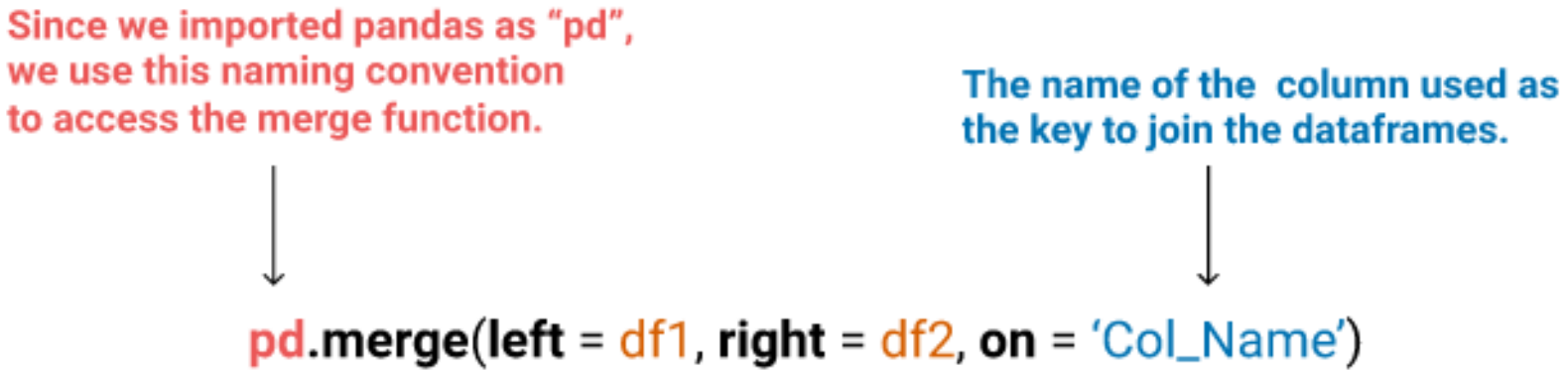
MERGE() FUNCTION

Table 8-2. merge function arguments

Argument	Description
`left`	DataFrame to be merged on the left side.
`right`	DataFrame to be merged on the right side.
`how`	One of `'inner'`, `'outer'`, `'left'`, or `'right'`; defaults to `'inner'`.
`on`	Column names to join on. Must be found in both DataFrame objects. If not specified and no other join keys given, will use the intersection of the column names in `left` and `right` as the join keys.
`left_on`	Columns in `left` DataFrame to use as join keys.
`right_on`	Analogous to `left_on` for `left` DataFrame.
`left_index`	Use row index in `left` as its join key (or keys, if a MultiIndex).
`right_index`	Analogous to `left_index`.
`sort`	Sort merged data lexicographically by join keys; `True` by default (disable to get better performance in some cases on large datasets).
`suffixes`	Tuple of string values to append to column names in case of overlap; defaults to `('_x', '_y')` (e.g., if `'data'` in both DataFrame objects, would appear as `'data_x'` and `'data_y'` in result).
`copy`	If `False`, avoid copying data into resulting data structure in some exceptional cases; by default always copies.
`indicator`	Adds a special column `_merge` that indicates the source of each row; values will be `'left_only'`, `'right_only'`, or `'both'` based on the origin of the joined data in each row.

MERGE() FUNCTION

- **Syntax**



MERGE() FUNCTION

- **Example**

df1

	orderID	TotalPrice
0	10002	1000
1	10003	5000
2	10004	12000
3	10005	7000

Left join →

df2

	orderID	ProductName
0	10002	Apples
1	10002	Grape
2	10003	WaterMelon
3	10003	Grape
4	10003	Pear
5	10004	Cherry
6	10005	Figue
7	10005	Mango

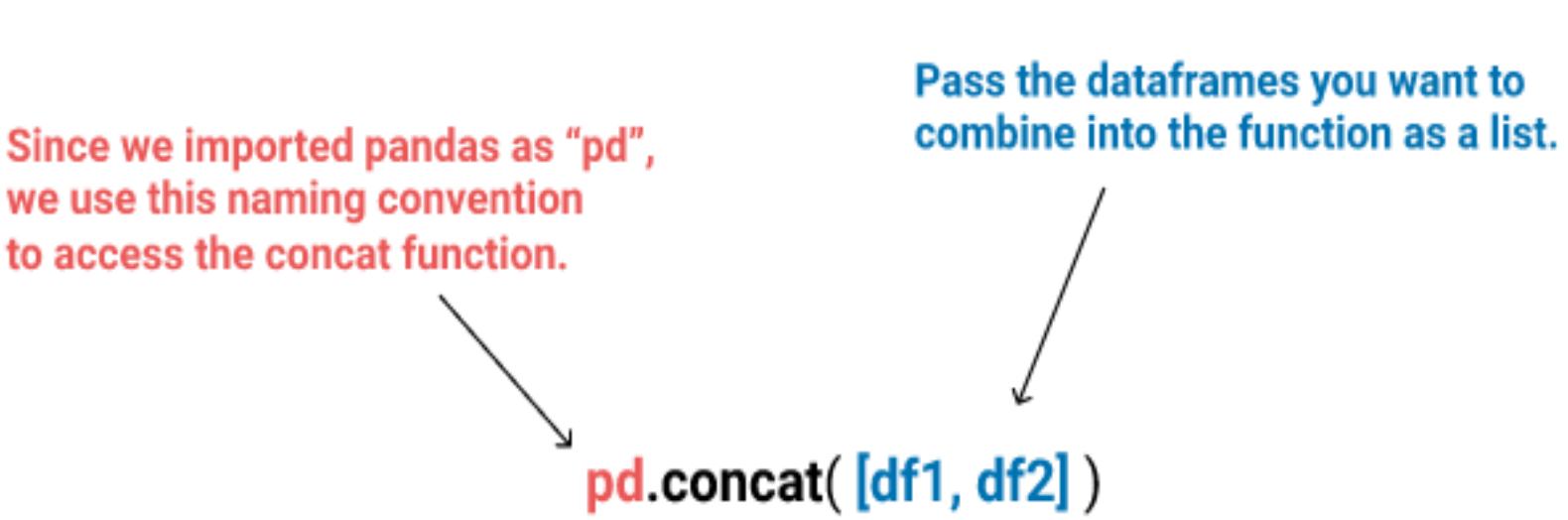
→

Result

	orderID	TotalPrice	ProductName
0	10002	1000	Apples
1	10002	1000	Grape
2	10003	5000	WaterMelon
3	10003	5000	Grape
4	10003	5000	Pear
5	10004	12000	Cherry
6	10005	7000	Figue
7	10005	7000	Mango

CONCAT() FUNCTION

- **Syntax**



CONCAT() FUNCTION

- The concat() function combines dataframes one of two ways

df1

	one	two
a	0	1
b	2	3
c	4	5

Axis=0

df2

	one	two
a	5	6
c	7	8

Concanated df

	one	two
a	0	1
b	2	3
c	4	5
a	5	6
c	7	8

- pd.concat([df1,df2])
- Rename the columns of df2 to [‘three’, ‘four’]
- Try concatenating df1 and df2 again and see what happens?

CONCAT() FUNCTION

- The concat() function combines dataframes one of two ways

df1

	one	two
a	0	1
b	2	3
c	4	5

df2

	three	four
a	5	6
c	7	8

Axis=1

Concanated df

	one	two	three	four
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

Pd.concat()

Argument	Description
`objs`	List or dict of pandas objects to be concatenated; this is the only required argument
`axis`	Axis to concatenate along; defaults to 0 (along rows)
`join`	Either `'inner'` or `'outer'` (`'outer'` by default); whether to intersection (inner) or union (outer) together indexes along the other axes
`join_axes`	Specific indexes to use for the other *n*–1 axes instead of performing union/intersection logic
`keys`	Values to associate with objects being concatenated, forming a hierarchical index along the concatenation axis; can either be a list or array of arbitrary values, an array of tuples, or a list of arrays (if multiple-level arrays passed in `levels`)

Argument	Description
`levels`	Specific indexes to use as hierarchical index level or levels if keys passed
`names`	Names for created hierarchical levels if `keys` and/or `levels` passed
`verify_integrity`	Check new axis in concatenated object for duplicates and raise exception if so; by default (`False`) allows duplicates
`ignore_index`	Do not preserve indexes along concatenation `axis`, instead producing a new `range(total_length)` index

Summary

	pd.concat()	pd.merge()
Default Join Type	Outer	Inner
Can Combine More Than Two Dataframes at a Time?	Yes	No
Can Combine Dataframes Vertically (axis=0) or Horizontally (axis=1)?	Both	Horizontally
Syntax	**Concat (Vertically)** concat([df1,df2,df3]) **Concat (Horizontally)** concat([df1,df2,df3], axis = 1)	**Merge (Join on Columns)** merge(left = df1, right = df2, how = 'join_type', on = 'Col') **Merge (Join on Index)** merge(left = df1, right = df2, how = 'join_type', left_index = True, right_index = True)

Project: RFM Analysis

Project Goals

Project Goals:

- Data aggregation
- Combining of data
- Basic Data Cleaning
- Discretization and Binning

Dataset description:

- This transactional data set contains all the transactions occurring between 2010 and 2011 for a UK-based and registered non-store online retail. The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers

RFM Analysis

RFM (Recency, Frequency, Monetary) analysis is a behavior-based approach to grouping customers into segments. It groups the customers based on their previous purchase transactions. How recently, how often, and how much did a customer buy? In this section, we will practice implementing the customer segment based on the RFM model.

RFM Analysis

- Frequency is the number of orders made by a customer.
- Recency is the number of days counting from the last date a customer makes an order to the last date in the analysis cycle. For example, if the last date in the analysis cycle is 30/11/2011, the last date a customer makes an order is 25/11/2011. As a result, the recency value is 5.
- Monetary is the total amount of money the customer has purchased during the analysis cycle

Attribute Information

- InvoiceNo: Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter 'c', it indicates a cancellation.
- StockCode: Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.
- Description: Product (item) name. Nominal.
- Quantity: The quantities of each product (item) per transaction. Numeric.
- InvoiceDate: Invoice Date and time. Numeric, the day and time when each transaction was generated.
- UnitPrice: Unit price. Numeric, Product price per unit in sterling.
- CustomerID: Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.
- Country: Country name. Nominal, the name of the country where each customer resides.

Discretization and Binning

Pd.cut()

Parameters	Descriptions
x	The input array to be binned. Must be 1-dimensional
bins	Int, sequence of scalars, or IntervalIndex • Defines the number of equal-width bins in the range of x. The range of x is extended by .1% on each side to include the minimum and maximum • Sequence of scalars: Defines the bin edges allowing for non-uniform width. No extension of the range of x is done • IntervalIndex: Defines the exact bins to be used. Note that IntervalIndex for bins must be non-overlapping
labels	Array or False, default None. Specifies the labels for the returned bins. Must be the same length as the resulting bins
precision	The precision at which to store and display the bins labels
duplicates	Default ‘raise’, ‘drop’. If bin edges are not unique, raise ValueError or drop non-uniques

Discretization and Binning

pd.qcut()

Parameters	Descriptions
x	The input array to be binned. Must be 1-dimensional
q	Int or list-like of float Number of quantiles. 10 for deciles, 4 for quartiles, etc. Alternately array of quantiles, e.g [0,.25,.5,.75,1.] for quartiles
labels	Array or False, default None. Specifies the labels for the returned bins. Must be the same length as the resulting bins
precision	The precision at which to store and display the bins labels
duplicates	Default ‘raise’, ‘drop’. If bin edges are not unique, raise ValueError or drop non-uniques

Cleaning Data

- Use the pd.read_csv() function to read the Online Retail.xlsx file into a dataframe called df
- Use the DataFrame.info() and DataFrame.head() methods to print information about both dataframes, as well as the first few rows
- Check duplicated rows
- Which attribute determines other attributes?

Cleaning Data

- Check the distribution of the *Quantity* attribute using the pd.describe() method
- What anomalies have you observed?
- Check %null value for each attribute
- Drop duplicated rows
- Delete records with the value of Quantity <=0
- Drop records with null CustomerID

RFM Analysis

- Calculate Recency, Frequency, and Monetary values for each customer. Store the result in a dataframe named the customer_df
- Create an empty DataFrame named customer_df
- Append a new column named CustomerID. This column stores the unique ID of each customer
- Create a new column named ‘TotalRevenue’ as the product of two columns Quantity and UnitPrice

RFM Analysis

- Create a new dataframe named frequency_df. This dataframe has only two columns:
  - The ‘CustomerID’ column stores the unique ID of each customer
  - The ‘Frequency’ column stores the frequency value of each customer
- Create a new dataframe named monetary_df. This dataframe has only two columns:
  - The ‘ CustomerID’ column stores the unique ID of each customer
  - The ‘Monetary’ column stores the total revenue of each customer
- Create a new dataframe named recency_df. This dataframe has only two columns:
  - The ‘CustomerID’ column stores the unique ID of each customer
  - The ‘Recency’ column stores the recency value of each customer

RFM Analysis

- Merge frequency_df, monetary_df, recency_df to customer_df dataframe such that this dataframe has four columns: CustomerID, Frequency, Monetary, Recency
- Add segment bin values to RFM table using quartile. For example, If the recency value belongs to the first quartile, the recency value will be replaced by 1. If it belongs to the second quartile, that value will be replaced by 2
  - Hint: using pd.qcut, create new columns named r_quantile, f_quantile, and m_quantile in the dataframe customer_df
  - In the customer_df, Create a new column named RFM_Score(1 point). The formula for RFM_Score is as follows

    RFM_Score = r_quantile + f_quantile + m_quantile

RFM Analysis

Based on RFM_Score, customers are divided into 3 segments: low-value, mid-value, and high-value so that it satisfies the following rules:

- The number of customers in the high-value segment does not exceed 20% of the total number of customers.
- The number of customers in the mid-value segment is no less than 30% of the total number of customers.