**Programming for Data Science**

**Data Munging with the pandas library**

**What you will learn / do in this lab**

1. How to use the pandas Python library to load data, extract subsets of data from it, transform the data and save the data

**Table of Contents**

[1. Overview 4](#_Toc43376231)

[A. What you will do for this lab 4](#_Toc43376232)

[B. Intro to pandas 4](#_Toc43376233)

[2. Create Pandas Series 5](#_Toc43376234)

[A. Create Pandas Series 5](#_Toc43376235)

[Create series with list and default index 5](#_Toc43376236)

[Create series with list and user-specified index 5](#_Toc43376237)

[Create a Pandas Series object with a dict object 5](#_Toc43376238)

[Create a Pandas Series object with datetime dtype 6](#_Toc43376239)

[3. Create Pandas DataFrames 7](#_Toc43376240)

[A. Create Pandas DataFrames 7](#_Toc43376241)

[Create DataFrame with Numpy array and default index 7](#_Toc43376242)

[Create DataFrame with Numpy array and user-specified index 7](#_Toc43376243)

[Create DataFrame with nested dict object 8](#_Toc43376244)

[Create DataFrame with dict of lists 8](#_Toc43376245)

[Create DataFrame with dict of Series object 8](#_Toc43376246)

[4. View data 10](#_Toc43376247)

[A. Understand shape, index, columns of dataset 11](#_Toc43376248)

[Task 1: Display the shape, index, columns of Series 11](#_Toc43376249)

[Use shape,index,values, count() on Series 11](#_Toc43376250)

[Task 2: Display the shape, index, columns of DataFrame 12](#_Toc43376251)

[Use shape,index,values, count() on DataFrame 12](#_Toc43376252)

[B. Retrieve statistical info of dataset 13](#_Toc43376253)

[Task 1: Display statistical information of Series and DataFrame 13](#_Toc43376254)

[Use sum,cumsum on Series 13](#_Toc43376255)

[Task 2: Display statistical information of Series and DataFrame 13](#_Toc43376256)

[Use sum,cumsum on DataFrame 13](#_Toc43376257)

[5. Loading and Saving Datasets 14](#_Toc43376258)

[A. Loading Data 14](#_Toc43376259)

[Task 1: Load data from CSV file 14](#_Toc43376260)

[Load CSV file into DataFrame 14](#_Toc43376261)

[Task 2: Load data from Excel spreadsheet 15](#_Toc43376262)

[Load Excel file into DataFrame 15](#_Toc43376263)

[B. Saving Data 16](#_Toc43376264)

[Task 1: Save DataFrame data to CSV and Excel 16](#_Toc43376265)

[Save DataFrame into CSV and Excel 16](#_Toc43376266)

[6. Selection 17](#_Toc43376267)

[A. Subset columns 17](#_Toc43376268)

[Task 1: Select column(s) by column name 17](#_Toc43376269)

[Pick specific years using their column names (loc) 17](#_Toc43376270)

[Task 2: Select columns by index 18](#_Toc43376271)

[Pick specific columns using their column index (iloc) 18](#_Toc43376272)

[Task 3: Create derived columns by boolean logic 19](#_Toc43376273)

[Pick specific columns with values > 500k 19](#_Toc43376274)

[Task 4: Select columns by regular expression 20](#_Toc43376275)

[Pick specific columns that start with characters 201 (filter) 20](#_Toc43376276)

[B. Subset rows 21](#_Toc43376277)

[Task 1: Select rows(s) by index label 21](#_Toc43376278)

[Use loc method to select rainfall data from rows containing 1982 only 21](#_Toc43376279)

[Task 2: Select row(s) by index 22](#_Toc43376280)

[Use iloc method to select rainfall data from last 12 rows of dataset 22](#_Toc43376281)

[Task 3: Select row(s) by boolean indexing 23](#_Toc43376282)

[Select rows that have > 300mm of rainfall 23](#_Toc43376283)

[7. Reshaping data 25](#_Toc43376284)

[A. Drop columns/rows from a DataFrame 25](#_Toc43376285)

[Task 1: Drop columns and rows from a DataFrame 25](#_Toc43376286)

[B. Append columns/rows to a DataFrame 27](#_Toc43376287)

[Task 1: Concatenate the data of two DataFrames by rows 27](#_Toc43376288)

[Concat data from two CSVs with performing arts and sports course info 27](#_Toc43376289)

[Task 2: Concatenate the data of two DataFrames by columns 28](#_Toc43376290)

[Concat school information data from two CSVs with union of columns 28](#_Toc43376291)

[C. Spread rows into columns (pivot) 29](#_Toc43376292)

[Task 1: Transpose rows into columns 29](#_Toc43376293)

[Pivot year data to columns 29](#_Toc43376294)

[D. Gather columns into rows (melt) 30](#_Toc43376295)

[Task 1: Gather columns into rows 30](#_Toc43376296)

[Unpivot year data from columns to rows 30](#_Toc43376297)

[E. Sort a DataFrame 31](#_Toc43376298)

[Task 1: Sort DataFrame by column values 31](#_Toc43376299)

[Sort data by column tax amount 31](#_Toc43376300)

[Task 2: Sort DataFrame by index values 33](#_Toc43376301)

[Sort data by index values 33](#_Toc43376302)

[F. Reindexing DataFrame 34](#_Toc43376303)

[Task 1: Reindex a DataFrame 34](#_Toc43376304)

[Reindex public transport data to add new years 34](#_Toc43376305)

[Task 2: Reset the index of a DataFrame 35](#_Toc43376306)

[Reset multi-index 35](#_Toc43376307)

[G. Rename the columns of a DataFrame 36](#_Toc43376308)

[Task 1: Rename the columns of a DataFrame 36](#_Toc43376309)

[Rename column from “type\_of\_public\_transport” to “Transport Type” 36](#_Toc43376310)

[8. Handling missing data 37](#_Toc43376311)

[A. Detect missing values with isnull()/ notnull() 37](#_Toc43376312)

[Task 1: Detect missing values with isnull() and notnull() 37](#_Toc43376313)

[Identify columns with missing values with isnull() 37](#_Toc43376314)

[B. Drop missing values with dropna 38](#_Toc43376315)

[Task 1: Drop missing values with dropna 38](#_Toc43376316)

[Drop missing values with dropna() 38](#_Toc43376317)

[C. Fill missing values (fillna) 39](#_Toc43376318)

[Task 1: Fill missing values with a scalar value 39](#_Toc43376319)

[Fill missing values with scalar value, forward fill, backward fill 39](#_Toc43376320)

[9. Combine datasets 41](#_Toc43376321)

[A. Merge datasets 41](#_Toc43376322)

[Task 1: Combine two datasets with merge method 41](#_Toc43376323)

[Combine students and their test scores 41](#_Toc43376324)

# Overview

## What you will do for this lab

In this lab, you will hone your skills in using the pandas library to load data, retrieve information about it, manipulate the data in different ways and visualize the data using MatplotLib.

The objective of this lab is to build up your familarity with the pandas library functions and how you can use it to perform basic data analysis.

## Intro to pandas

pandas (<https://pandas.pydata.org/>) is an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Python has long been great for data munging and preparation, but less so for data analysis and modeling. pandas helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R.

Below are the main highlights of the pandas library:-

* Provides fast and efficient DataFrame object for data manipulation with integrated indexing;
* Provides tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
* Provides a plethora of functions to handle:
  + Slice / subset larget datasets
  + Reshape and pivot datasets
  + Handle missing data, lexible reshaping and pivoting of data sets;
  + Insert / delete column or row data
  + high performance merging and joining of data sets;
* Provides useful functions to handle time series data:: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.

Pandas is highly optimized for performance, with critical code paths written in Cython or C.

Python with pandas is in use in a wide variety of academic and commercial domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

However, do note that pandas does not implement significant modeling functionality outside of linear and panel regression; for this, look to statsmodels and scikit-learn

# Create Pandas Series

* Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects etc)
* The axis labels are collectively referred to as the index

The basic method to create a Pandas Series is as follows:-



* *data* can be Python lists, Python dictionary objects, Numpy arrays or scalar values
* *index* is the axis labels and can have default values or being explicitly defined

Demonstrate the different ways to create Panda series objects by completing the questions below.

## Create Pandas Series

|  |  |
| --- | --- |
|  | Create series with list and default index Create and output a Pandas Series object with default index, containing values as shown on the right |
|  | Create series with list and user-specified index Create and output a Pandas Series object with a specified index (a to f) as shown on the right |
|  | Create a Pandas Series object with a dict object Create a Pandas Series object and output as shown on the right.  Hint: Use a dict object  What is the index in this Series object? |
|  | Create a Pandas Series object with datetime dtype Create a Pandas Series object and output as shown below.    Hint: use pd.date\_range |

# Create Pandas DataFrames

* **DataFrame** is the most commonly used data structure in pandas
* DataFrame is a **2-dimensional** labeled data structure
* You can think of it like an Excel spreadsheet with rows and columns

The basic method to create a Pandas DataFrame is as follows:-



* ***data*** : numpy ndarray, dict, Series or DataFrame
* ***index*** : Index to use for resulting frame. Default to np.arange(n)
* ***columns***  : Column labels to use for resulting frame. Default to np.arange(n)

Demonstrate the different ways to create Panda DataFrame objects by completing the questions below.

## Create Pandas DataFrames

|  |  |  |
| --- | --- | --- |
|  | Create DataFrame with Numpy array and default index Create a Pandas DataFrame object with default index, containing a 5x5 Numpy array of random numbers between 100 and 200 as shown below | |
|  | Create DataFrame with Numpy array and user-specified index Modify the code for part (a) such that the DataFrame object is now mapped to the index and columns as shown below | |
|  | Create DataFrame with nested dict object Given the following dict object uni\_graduates, create a Pandas DataFrame object and print out its contents and type as shown below.  uni\_graduates = {  2010:{'Females':6300,'Males':6496}, 2011:{'Females':7281,'Males':6428},  2012:{'Females':7114,'Males':6736}, 2013:{'Females':8170,'Males':7785},  2014:{'Females':7620,'Males':7756}  } | |
|  | Create DataFrame with dict of lists Create a Pandas DataFrame object with the dict of list nus\_median\_gross\_pay\_2013to2015 and print out its contents and type as shown below.  nus\_median\_gross\_pay\_2013to2015 = {  'BA' : [3000,3000,3000],  'BA (Honours)' : [3200,3520,3800],  'BA (Accountancy)' : [2700,2838,2850],  'BA (Accountancy) (Honours)' : [2800,2912,3000]  }   |  | | --- | |  | | |
|  | Create DataFrame with dict of Series object Create a Pandas DataFrame object with the dict of Series variable monthswith31days and print out its contents and type as shown below.  jan = pd.date\_range('20170101', periods=31)  mar = pd.date\_range('20170301', periods=31)  may = pd.date\_range('20170501', periods=31)  jul = pd.date\_range('20170701', periods=31)  aug = pd.date\_range('20170801', periods=31)  oct = pd.date\_range('20171001', periods=31)  dec = pd.date\_range('20171201', periods=31)  monthswith31days = {"Jan": jan, "Mar": mar,"May": may, "Jul": jul,"Aug": aug, "Oct": oct, "Dec": dec}    …. | |
|  | Create the Pandas DataFrame object as shown below, by passing a numpy array, with a datetime index and labeled columns | |  | | --- | | import numpy as np  import pandas as pd  npArray = np.random.randn(6,4) | |

# View data

Create the following two pandas objects:

|  |  |
| --- | --- |
|  | a Pandas Series object ***myseries***with default index containing 1000 random values between 0 and 1 |
|  | a Pandas DataFrame object ***mydf*** with default index containing 1000 rows of data that has 3 columns per row, where each column of data contains values as specified below:   * Column A: Random numeric values between 10 and 100 * Column B: The value zero * Column C: stepping values from 0 to 999 |

Answer the questions in section 4A to 4B using these two pandas objects you have created.

## Understand shape, index, columns of dataset

|  |  |  |
| --- | --- | --- |
| **shape** | Returns the axis dimensions (rows, columns) of the object | myseries.shape  mydf.shape |
| **index** | Describes the index | myseries.index  mydf.index |
| **columns** | Describe the columns of a DataFrame | mydf.columns |
| **values** | Return the values of the Series or DataFrame as a np array | myseries.  mydf.values |
|  |  |  |
| **count()** | Returns the number of non NA values | mydf.count() |
| **info()** | Provides information on the DataFrame | mydf.info() |
| **describe()** | Show a quick statistic summary of your data | mydf.describe() |

### Task 1: Display the shape, index, columns of Series

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Use shape,index,values, count() on Series Use the shape, index, values attributes as well as the count() function to retrieve more information about ***myseries*** as follows:-   |  |  | | --- | --- | | **Expected output from myseries.shape** | **Expected output from myseries.index** | |  |  |  |  | | --- | | **Expected output from myseries.values** | |  |  |  |  | | --- | --- | | **Expected output from myseries.count()** | **Expected output from myseries.describe()** | |  |  | |

### Task 2: Display the shape, index, columns of DataFrame

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Use shape,index,values, count() on DataFrame Use the shape, index, columns and values attributes as well as the count() and info() functions to retrieve more information about *mydf*.   |  |  | | --- | --- | | **Expected output from mydf.shape** | **Expected output from mydf.index** | |  |  |  |  |  | | --- | --- | | **Expected output from mydf.columns** | **Expected output from mydf.count()** | |  |  |  |  |  | | --- | --- | | **Expected output from mydf.values** | **Expected output from mydf.info()** | |  |  |  |  | | --- | | **Expected output from mydf.describe()** | |  | |

## Retrieve statistical info of dataset

|  |  |  |
| --- | --- | --- |
| [sum](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.sum.html) | Return the sum of the values for the requested axis | df.sum() |
| [cumsum](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.cumsum.html) | Return cumulative sum over requested axis | myseries.cumsum() mydf.cumsum() |
| [min](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.min.html) | Minimum / Maximum values | mydf.min(axis=None, skipna=None, level=None, numeric\_only=None, \*\*kwargs) |
| **max** |  |  |
| **idxmin** |  | df.idxmin() |
| [idxmax](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.idxmax.html) | Return index of first occurrence of maximum over requested axis. NA/null values are excluded. | df.idxmax(axis=0, skipna=True) |
|  |  |  |
| [mean](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.mean.html) | Mean of values | df.mean(axis=None, skipna=None, level=None, numeric\_only=None, \*\*kwargs) |
| **median** | Median of values | df.median() |
| **describe()** | Show a quick statistic summary of your data | mydf.describe() |

### Task 1: Display statistical information of Series and DataFrame

|  |  |
| --- | --- |
|  | Use sum,cumsum on Series Use all of the above methods to retrieve summary statistical information about ***myseries.*** Format your output similar to Figure below. |

|  |
| --- |
|  |

### Task 2: Display statistical information of Series and DataFrame

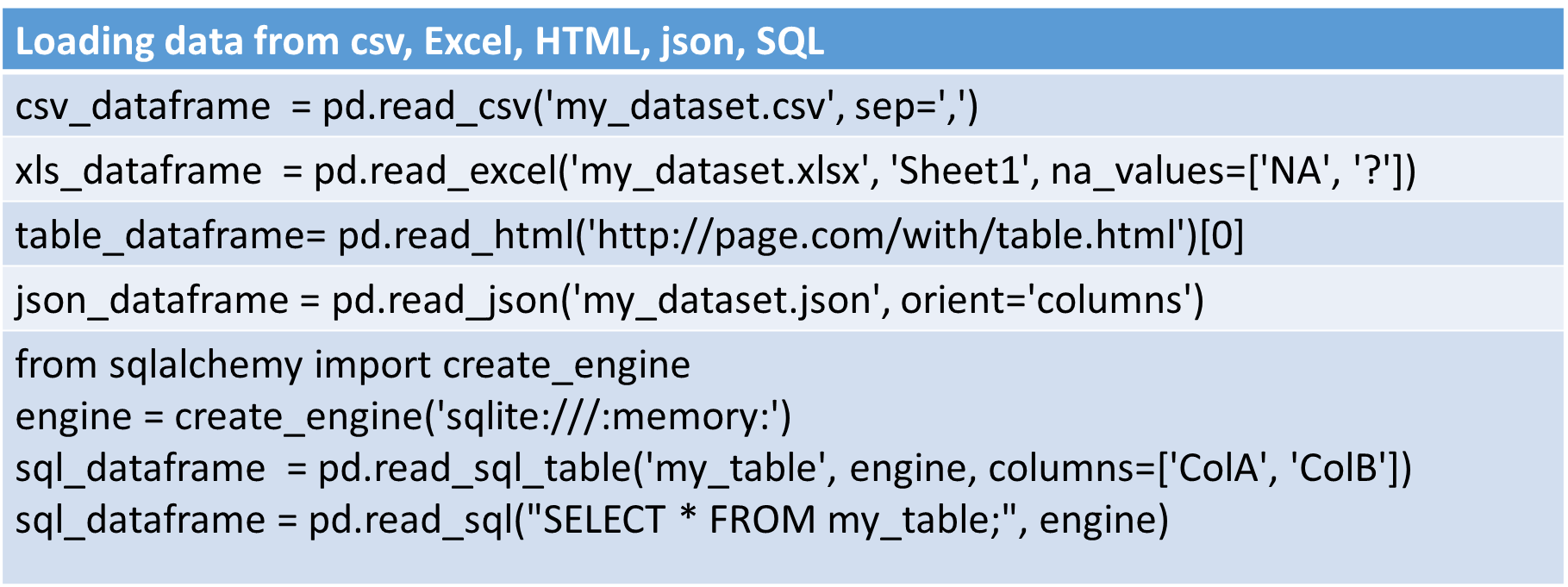
|  |  |
| --- | --- |
|  | Use sum,cumsum on DataFrame Use all of the above methods to retrieve summary statistical information about ***mydf.***  Format your output similar to Figure above. |

# Loading and Saving Datasets

Pandas provides methods to load and save data from a variety of sources.

Practise how to load and save data from sources such as CSV files, Excel files, HTML pages and json data by completing the exercises in this section.

## Loading Data



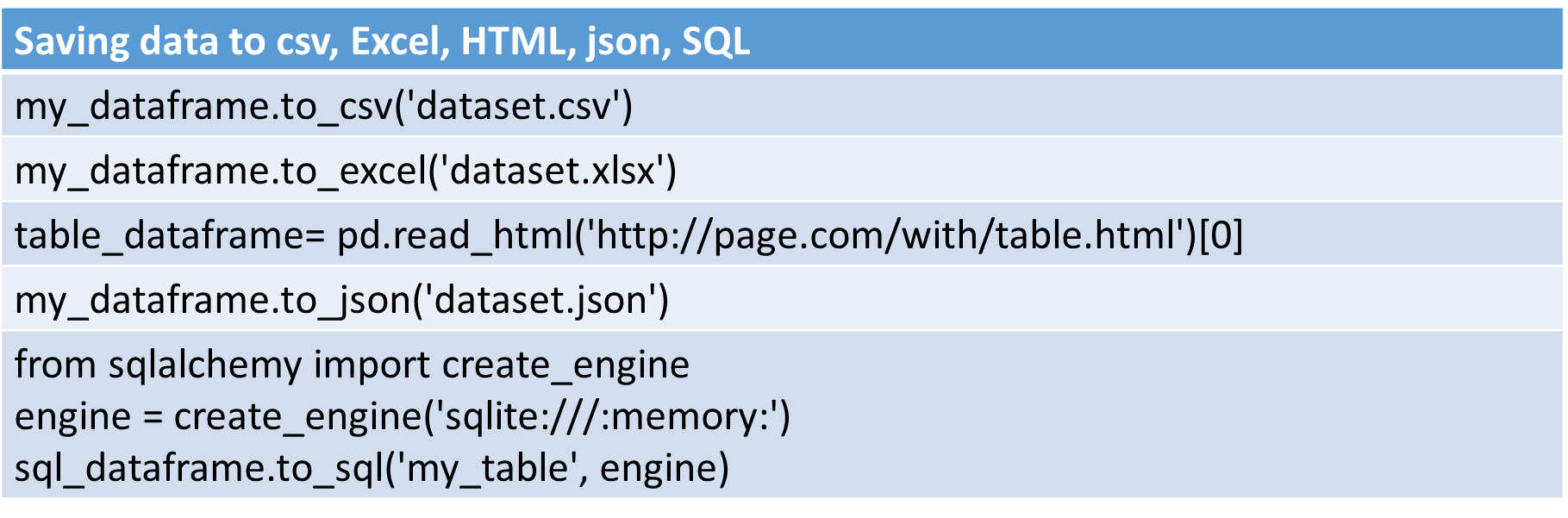
### Task 1: Load data from CSV file

|  |  |
| --- | --- |
|  | Load CSV file into DataFrame Create a Pandas DataFrame object *df\_csv* with default index by loading the data from the ***income.csv*** file. Note that the columns in this file are delimited by a semi-colon character.   * How many rows and columns does *df\_csv* contain? * What are the datatypes in df\_csv? |
|  | |

### Task 2: Load data from Excel spreadsheet

|  |  |
| --- | --- |
|  | Load Excel file into DataFrame Create a Pandas DataFrame object *df\_xls* with default index by loading the data from the ***singstats\_household.xlsx*** file.   * How many rows and columns does *df\_xls* contain? * What are the datatypes in df\_xls? |
|  | |

## Saving Data



### Task 1: Save DataFrame data to CSV and Excel

|  |  |  |
| --- | --- | --- |
|  | Save DataFrame into CSV and Excel Create a Pandas DataFrame object with the index columns and column names as shown, containing a 5x5 numpy array of random numbers between 100 and 200 as shown.  Save the dataset to a CSV file “data/mydf.csv” as well as a Excel spreadsheet “data/mydf.xlsx” |  |

# Selection

pandas provides optimized data access methods, .at, .iat, .loc,.iloc and .ix for selecting subsets of a dataset. Practise these methods by completing the lab questions below.

## Subset columns

### Task 1: Select column(s) by column name

|  |  |
| --- | --- |
|  | Pick specific years using their column names (loc) Load the data from the ***singstats\_maritalstatus.xlsx*** file into a Pandas DataFrame object mydf . Set the index as the first column of the Excel file (i.e. “Variables”)   * Select all the rows in the 1986 column and store them in a dataframe named df\_1986***.*** Print out the values of df\_1986 * Select all the rows in the 1980 to 1989 columns and store them in a dataframe named df\_1980s. Print out the values of df\_1980s   Your output should look similar to that below |

### Task 2: Select columns by index

|  |  |
| --- | --- |
|  | Pick specific columns using their column index (iloc) Load the data from the ***singstats\_maritalstatus\_2006\_to\_2016.xlsx*** file into a Pandas DataFrame object mydf . Set the index as the first column of the Excel file (i.e. “Variables”)  • Select the second,third and fourth columns of data and store them in a dataframe named df\_index123***.*** Print out the values of df\_index123   * Select the last 5 columns and store them in a dataframe named df\_last5cols***.*** Print out the values of df\_last5cols   Your output should look similar to that below |

### Task 3: Create derived columns by boolean logic

|  |  |
| --- | --- |
|  | Pick specific columns with values > 500k Load the data from the ***singstats\_maritalstatus.xlsx*** file into a Pandas DataFrame object ***mydf*** . Set the index as the first column of the Excel file (i.e. “Variables”)  • Find out how many rows of the data in the 1980 column have values more than 500000 and how many rows by of the data have values less than or equal to 500000 by creating a derived column using boolean indexing on this column and using the count() method you have learnt in Section 4 of this lab.  Your output should look similar to that below. |

### Task 4: Select columns by regular expression

|  |  |
| --- | --- |
|  | Pick specific columns that start with characters 201 (filter) Load the data from the ***singstats\_maritalstatus.xlsx*** file into a Pandas DataFrame object ***mydf*** . Set the index as the first column of the Excel file (i.e. “Variables”)   * Select the columns that start with “201” and store them in a dataframe named df\_2010\_and\_after * Print out the values of df\_2010\_and\_after   Your output should look similar to that below. |

## Subset rows

### Task 1: Select rows(s) by index label

|  |  |
| --- | --- |
|  | Use loc method to select rainfall data from rows containing 1982 only Load the data from the *rainfall-monthly-total.csv* file into a Pandas DataFrame object ***df\_rainfall***. Set the index as the first column of the file (i.e. “month”)  Use ***df\_rainfall.loc*** to select only rows from 1982 and display the data using Matplotlib as shown.  A close up of a logo  Description generated with very high confidence  Figure 1: Rainfall data in 1982 from data.gov.sg |

### Task 2: Select row(s) by index

|  |  |
| --- | --- |
|  | Use iloc method to select rainfall data from last 12 rows of dataset Load the data from the *rainfall-monthly-total.csv* file into a Pandas DataFrame object ***df\_rainfall***.  Set the index as the first column of the file (i.e. “month”)  Use ***df\_rainfall.iloc*** to select the last 12 rows of the dataset and display the data using Matplotlib as shown.    Figure 2: The last 12 months of data from data.gov.sg |

### Task 3: Select row(s) by boolean indexing

|  |  |
| --- | --- |
|  | Select rows that have > 300mm of rainfall Load the data from the *rainfall-monthly-total.csv* file into a Pandas DataFrame object ***df\_rainfall***.  Set the index as the first column of the file (i.e. “month”)  Use boolean indexing to select only the rows of data that has more than 300 mm of rainfall and sort the resulting data in ascending order.  Display the twelve year/month with the most rainfall in a barchart using Matplotlib as shown. |
|  | |

Task 4: Select unique rows only

|  |  |
| --- | --- |
|  | Load the data from the *rainfall-monthly-total-withduplicates.csv* file into a Pandas DataFrame object ***df\_rainfall***.  Set the index as the first column of the file (i.e. “month”)  Use the drop\_duplicates method to ensure you select only unique rows in the dataset.  Sort the resulting data in descending order.  Display the twelve year/month in a barchart using Matplotlib as shown. |
| C:\Users\Dora Chua\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\166BEC42.tmp | |

# Reshaping data

pandas provides optimized data access methods, .at, .iat, .loc,.iloc and .ix for selecting subsets of a dataset.

Practise these methods by completing the lab questions below.

## Drop columns/rows from a DataFrame

### Task 1: Drop columns and rows from a DataFrame

|  |  |
| --- | --- |
|  | Load the data from the graduate-employment-survey-ntu-nus-sit-smu-sutd-utf8.csv file into a Pandas DataFrame object ***df*** .  Using the **drop()** method, perform the following operations in order and print out the shape of the transofrmed DataFrame object at ech stage.  Your output should resemble that in the screenshot below.  **Operation 1**  Remove the following columns:-   * School * employment\_rate\_overall * employment\_rate\_permanent * basic\_monthly\_mean * basic\_monthly\_median * gross\_mthly\_25\_percentile * gross\_mthly\_75\_percentile   **Operation 2**  Remove data that contains the following names in the ‘university’ column   * "Nanyang Technological University" * "Singapore Institute of Technology" * "Singapore University of Technology and Design"   **Operation 3**  Remove data that contains the following names in the ‘year’ column   * 2013 * 2014   **Operation 4**  Remove data that is less than 4000 in the ‘gross\_monthly\_median’ column |
|  | |

## Append columns/rows to a DataFrame

### Task 1: Concatenate the data of two DataFrames by rows

|  |  |
| --- | --- |
|  | Concat data from two CSVs with performing arts and sports course info Load the data from the pa-performing-arts-courses.csv and pa-sports-courses.csv files into the Pandas DataFrame object df\_arts and df\_sports respectively   * Print out the shape of df\_arts as well as the of its first 2 rows and last 2 rows * Print out the shape of df\_sports as well as the data of its first 2 rows and last 2 rows * Use the concat method to combine the two DataFrame objects to a new DataFrame object named df\_all * Print out the shape of df\_all as well as the data of its first 3 rows and last 3 rows   Your output should look similar to that below. |

### Task 2: Concatenate the data of two DataFrames by columns

|  |  |
| --- | --- |
|  | Concat school information data from two CSVs with union of columns Load the data from the schoolinfo\_1a.csv and schoolinfo\_2a.csv files into the Pandas DataFrame object df\_1 and df\_2 respectively, with index column set to “school\_name”   * Print out the column names and the first 2 rows of df\_1 and df\_2 * Use the concat method to combine the two DataFrame objects to a new DataFrame object named df\_all * Print df\_all   Your output should look similar to that below |
|  | |

## Spread rows into columns (pivot)

### Task 1: Transpose rows into columns

|  |  |
| --- | --- |
|  | Pivot year data to columns Load the data from the irascollectionbytaxtype.csv into a Pandas DataFrame mydf.   * Print out the first 5 rows of mydf. You should see that each financial\_year e.g. 2002 is printed a few times for each tax\_type. * Use the pivot method to spread the financial\_year to be displayed on the column instead, and set tax\_type as the index column   Your output should look similar to that below |

## Gather columns into rows (melt)

### Task 1: Gather columns into rows

|  |  |
| --- | --- |
|  | Unpivot year data from columns to rows Load the data from the irascollectionbytaxtypemelt.csv into a Pandas DataFrame mydf.  • Print out the first 5 rows of mydf. You should see the output similar to that as shown in the first diagram below.  • Use the melt method to gather the column data by year into rows, then print out the transformed DataFrame. Your output should look similar to the second diagram below. |

## Sort a DataFrame

### Task 1: Sort DataFrame by column values

|  |  |
| --- | --- |
|  | Sort data by column tax amount Load the data from the irascollectionbytaxtype.csv file into a Pandas DataFrame object mydf.   * Print out the first 10 rows of mydf * Sort the data by the column tax\_collected in descending order * Print out the first 10 rows of the sorted dataframe * Print out the last 10 rows of the sorted dataframe   Your output should look similar to that below |

### Task 2: Sort DataFrame by index values

|  |  |
| --- | --- |
|  | Sort data by index values Load the data from the irascollectionbytaxtype.csv file into a Pandas DataFrame object mydf.   * Set the index columns to be “financial\_year” and “tax\_collected” * Print out the first 10 rows of mydf * Use the sort\_index method on the “tax\_collected” index to sort the data in descending order, ie. Highest tax amount first * Print out the first 5 rows of the sorted dataframe * Print out the last 5 rows of the sorted dataframe   Your output should look similar to that below |

## Reindexing DataFrame

### Task 1: Reindex a DataFrame

|  |  |
| --- | --- |
|  | Reindex public transport data to add new years Load the data from the publictransportjourney.csv into a Pandas DataFrame mydf. Set the index column as “year”   * Print out mydf * Create a new index mynewindex with the following values:   2000,2001,2002,2003,2004,2005,2006,2007,2008,2009,2010   * Use the ***reindex*** method to reindex mydf with mynewindex * Print out the reindexed mydf   Your output should look similar to that below |

### Task 2: Reset the index of a DataFrame

|  |  |
| --- | --- |
|  | Reset multi-index Load the data from the publictransport.csv into a Pandas DataFrame mydf.   * Use the ***set\_index*** method to set the index as “year” and “type\_of\_public\_transport”. Print out the first 10 rows of the dataset * Next, use the pandas ***reset\_index*** method to reset the index set earlier. Print out the first 10 rows of the dataset with the reset index   Your output should look similar to that below |

## Rename the columns of a DataFrame

### Task 1: Rename the columns of a DataFrame

|  |  |
| --- | --- |
|  | Rename column from “type\_of\_public\_transport” to “Transport Type” Load the data from the publictransport.csv file into a Pandas DataFrame object mydf .   * Print out the first 5 rows of the dataset * Next, use the pandas ***rename*** method to rename the column names to “Year”, “Transport Type” and “Average Ridership” respectively. * Print out renamed dataset   Your output should look similar to that below |

# Handling missing data

pandas provides various methods handling missing data: isnull, notnull, dropna and fillna.

Practise these methods by completing the lab questions below.

## Detect missing values with isnull()/ notnull()

### Task 1: Detect missing values with isnull() and notnull()

|  |  |
| --- | --- |
|  | Identify columns with missing values with isnull() Load the data from the ***singstats\_maritalstatus.xlsx*** file into a Pandas DataFrame object ***mydf*** . Set the index as the first column of the Excel file (i.e. “Variables”) and consider values that have the value “-“ as being missing or invalid.   * Use the pandas isnull method to identify which of the columns in mydf contain missing or invalid data and store the output to a variable named df\_missing. Print out df\_missing.   Your output should look similar to that below |
|  |  |

## Drop missing values with dropna

### Task 1: Drop missing values with dropna

|  |  |
| --- | --- |
|  | Drop missing values with dropna() Load the data from the singstats\_maritalstatus.xlsx file into a Pandas DataFrame object mydf. Set the index as the first column of the Excel file (i.e. “Variables”) and consider values that have the value “-“ as being missing or invalid.   * Print out the first 10 rows of the dataset which should already reveal the columns or rows with missing or invalid data * Next, use the pandas ***dropna*** method to drop all the columns with missing data from mydf   Your output should look similar to that below |

## Fill missing values (fillna)

### Task 1: Fill missing values with a scalar value

|  |  |
| --- | --- |
|  | Fill missing values with scalar value, forward fill, backward fill Load the data from the hdbresaleindex\_missing.csv file into a Pandas DataFrame object mydf. Set the index as the first column of the csv file (i.e. “quarter”)   * Use the pandas **isnull()** method to identify which rows in mydf contains missing or invalid data. Store the a variable named df\_missing. Print out df\_missing. * Next, print out the result after using the pandas ***fillna()*** method to fill the missing values in mydf using the scalar value 50.0. * Next, print out the result after using the pandas ***fillna()*** method to fill the missing values in mydf using the ***forward fill***. * Next, print out the result after using the pandas ***fillna()*** method to fill the missing values in mydf using the ***backward fill***. * Next, print out the result after using the pandas ***fillna()*** method to fill the missing values in mydf using the ***mean of the ‘index’ column.***   Your output should look similar to that below |
|  |  |

# Combine datasets

For those with a background in databases and SQL, you will appreciate that pandas also supports join operations that are idiomatically very similar to SQL.

To perform a SQL like join operation in pandas, you can use the *merge* function in pandas.

|  |
| --- |
| pd.merge(left, right, how='inner', on=None, left\_on=None, right\_on=None, left\_index=False, right\_index=False,   sort=True, suffixes=('\_x', '\_y'), copy=True, indicator=False) |

## Merge datasets

### Task 1: Combine two datasets with merge method

|  |  |
| --- | --- |
|  | Combine students and their test scores  * Load the data from ***merge\_students-male.csv***, ***merge\_students-female.csv***, ***merge\_testscores.csv***  into the Pandas DataFrame objects df\_m,df\_fanddf\_testrespectively. * Combine df\_m and df\_f into a single dataset df\_students using the concat method * Merge df\_students and df\_test using the studentid value (i.e. INNER JOIN) into a dataset df\_merged * Print out the values of df\_merged, sorted by test scores, in descending order. Your output should look similar to that below. |

**-- End of Lab --**