

**Pandas Package**

**Introduction**

During this lab, you will learn about the Python data analysis with pandas package.

**Estimated Time**

30 minutes

**Objectives**

At the end of this lab, you will be able to:

* Get prepared to use the pandas package in Python code.
* Use data structures provided by pandas.
* Do data processing combining pandas and NumPy.

**Logon Information**

Use the following credentials to sign into virtual environment.

* Username: **Administrator**
* Password: **Passw0rd!**

Table of Contents

[Lab: Pandas Package 3](#_Toc521940005)

[Exercise 1: Data cleansing 3](#_Toc521940006)

Lab: Pandas Package

During this lab, you will learn about the Python data analysis with pandas package.

Exercise 1: Data cleansing

Tasks

1. Start Anaconda Prompt

Start Anaconda Prompt as an administrator from the **Start Menu**. If you can't find it in the Start Menu, search for it by typing **Anaconda** in the search bar. Then, right-click Anaconda Prompt and select **Run as administrator**.

1. Install pandas package

Python packages need to be installed for each virtual environment. For this lab, since we are going to use the conda base environment, Pandas is **already installed** by default. **If you use** **another virtual environment**, you should run the following command in Anaconda prompt to install Pandas.

conda install pandas

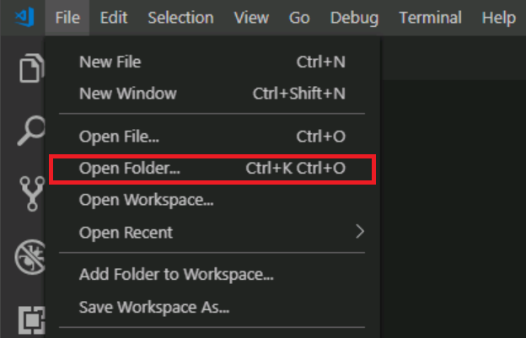
1. Start Visual Studio Code

Start Visual Studio Code from the **Start Menu**. If you can't find it in the Start Menu, search for it by typing **Visual** in the search bar.

When VS Code starts, you'll see a screen like the following:

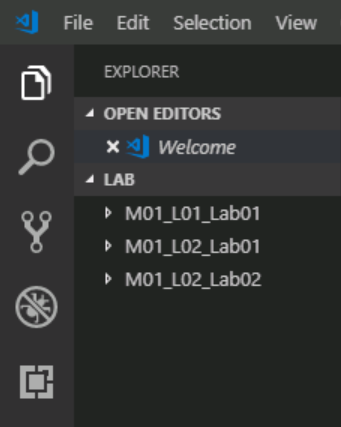
If you have used VS code in another task before, the VS code will start with the workspace folder opened. In that case, select **Close Folder** from the File menu.

1. Open Folder
2. Open the Lab folder from the File menu. Choose the File menu and select **Open folder**.



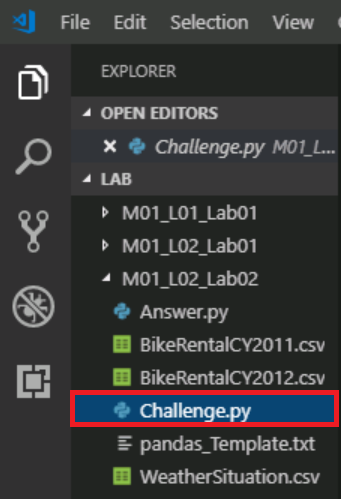
1. In the dialog box that appears, type the folder path as shown below:

C:\Labs\Module1\LAB



1. Open File

Select the **M01\_L02\_Lab02** folder on the side bar and select **Challenge.py**. This file is a template for a series of analysis.



1. Loading data #1

You were contracted to work to analyze the rental performance of a bike sharing service. This job has a predecessor, and the code to load the 2011 data has already been created. (Please check the data loading part of **Challenge**.**py**)

Please note that the answers for the challenge are in **Answer.py** file.

###############  
# Data preparation for 2011  
###############  
import numpy as np  
import pandas as pd  
  
df\_2011 = pd.read\_csv('C:\\Labs\\Module1\\LAB\\M01\_L02\_Lab02\\BikeRentalCY2011.csv')  
df\_2011.set\_index(['instant'], inplace=True)

You are going to see if the data is loaded correctly. Run the above code sequentially on VS code to see how the data is loaded.

pd.read\_csv() function can load csv formatted file easily.

If the data is successfully loaded, a DataFrame like the following is stored in memory:

dteday ... registered  
instant ...  
1 2011/1/1 ... 13  
2 2011/1/1 ... 32  
3 2011/1/1 ... 27  
4 2011/1/1 ... 10  
5 2011/1/1 ... 1  
... ... ... ...  
8644 2011/12/31 ... 36  
8645 2011/12/31 ... 27

1. Loading data #2

Your first task in this analysis is to write code to load the data for 2012. You may not be familiar with the data loading process, but fortunately the code left by your predecessor will be available as a sample.

Load the data for 2012. After the load, the data for 2011 should be combined with the 2011 data and stored in a single DataFrame.

You can use pd.concat() function or append() method to join the DataFrame row direction.

If you can properly load data and merge DataFrames, the following data is stored in memory:

dteday ... registered  
instant ...  
1 2011/1/1 ... 13  
2 2011/1/1 ... 32  
3 2011/1/1 ... 27  
4 2011/1/1 ... 10  
5 2011/1/1 ... 1  
... ... ... ...  
17378 2012/12/31 ... 48  
17379 2012/12/31 ... 37

Hint – You can use the following code. You still need to do a few changes.

df\_2012 = pd.read\_csv('**<file location>**\\BikeRentalCY2012.csv')

df\_2012.set\_index(**<same parameters from df\_2011.set\_index() >**)

df = df\_2011.append(df\_2012)

1. Joining DataFrame

You wanted to see the data in detail, but you couldn't understand the meaning of the data because the weathersit item is a number. Therefore, you decided to combine the master data to see what each value meant.

Load the weather master data. And, join the Dataframe you created in the previous task and weather master data. Finally, remove the **code** column for the master data because it is not necessary.

If the data is properly processed, a DataFrame with data columns similar to the following is created:

['dteday', 'season', 'yr', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'situation']

Hint – You can use the following code. You still need to do a few changes.

weather = pd.read\_csv('**<file location>**\\WeatherSituation.csv')

data = df.merge(weather, left\_on='weathersit', right\_on='code')

data.drop('**<column to remove>**', axis=1, inplace=True)

data.head()

1. Clean missing data

You start to see the details of the data and are concerned about missing values immediately. You want to check for missing values before you check the entire data, and if there are missing values, you can handle them appropriately.

For the DataFrame created in the previous task, check for missing values for each column. If you detect missing values, refer to the other data to fill in the missing values.

To check if some data in the DataFrame is missing, the isnull() method is available. You can also use any() method to determine whether one True exists in the set.

You can use the following code. Run each line to see the results.

data.isnull().any()

data[data['season'].isnull()]

data[data['dteday'] == '2011/1/2'].head()

data.loc[data['season'].isnull(), 'season'] = 1

data.isnull().any()

1. Processing Data

After you have properly processed the missing values, you have proceeded to verify the data. As a result, you realized that you need to create a data set as a sum of unregistered users (casual) and registered users (registered). You also need a percentage of unregistered users over the total.

Add a value that sums the casual and registered columns as **cnt** column. Also, add an unregistered user ratio as a **casual ratio** column.

Dataframe can easily handle operations between columns.

Use the following code as a guide. Remember that you have all code in the **Answer.py** file.

data['cnt'] = **<casual column>** + **<registered column>**

data['casual ratio'] = **<casual column>** / **<cnt column>**

1. Calculate basic statistics

As you have calculated in the previous analysis, you have decided to calculate the basic statistics to help you understand the summary of the data. However, you have decided to calculate statistics only for **temp**, **hum**, **windspeed**, **casual**, **registered** and **casual** **ratio** columns because some data columns do not provide a meaningful number.

Use the methods provided by pandas to calculate the basic statistics.

The describe() method computes the basic statistics for a column in the DataFrame as its summary.

If you calculate properly, you will get the following results:

temp ... casual ratio  
count 17379.000000 ... 17379.000000  
mean 0.496987 ... 0.172143  
std 0.192556 ... 0.136557  
min 0.020000 ... 0.000000  
25% 0.340000 ... 0.063492  
50% 0.500000 ... 0.146893  
75% 0.660000 ... 0.253731  
max 1.000000 ... 1.000000

Use the following code as a guide.

data[['column1', ' column2', ' column3', … ]].describe()

1. Aggregation

You are requested to aggregate by day, month of each year, and time from the customer. Therefore, you decided to consider the summary of the **cnt** column by the aggregate and the basic statistics first.

Aggregate the **cnt** columns by day, month of each year, and time units. For basic statistics, calculate the total, minimum, maximum, and standard deviation.

If you want to summarize on a monthly basis for each year, use multiple data columns.

If you calculate properly, the monthly aggregate results for each year are as follows:

sum min max std  
yr mnth  
0 1 38189 1 249 48.620653  
 2 48215 1 327 63.626005  
 ... ... ... ... ...  
 11 102167 1 553 123.748860  
 12 87323 1 477 106.943349  
1 1 96744 1 559 119.796712  
 2 103137 1 610 134.286493  
 ... ... ... ... ...

You can use the following code:

data.groupby(['dteday']).aggregate(['sum', 'min','max', np.std])['cnt']

data.groupby(['hr']).aggregate(['sum', 'min','max', np.std])['cnt']

data.groupby(['yr', 'mnth']).aggregate(['sum', 'min','max', np.std])['cnt']

Exercise 1 has been completed.