BA-BEAD Big Data Engineering for Analytics

Workshop Series

Spark DataFrame Basics

Table of Contents

List of python scripts and datasets

Scripts	Datasets
PythonDemo.py	NA
BasicStatistics.py	people.json
groupByAgg.py	sales_info.csv
missingdata.py	ContainsNull.csv

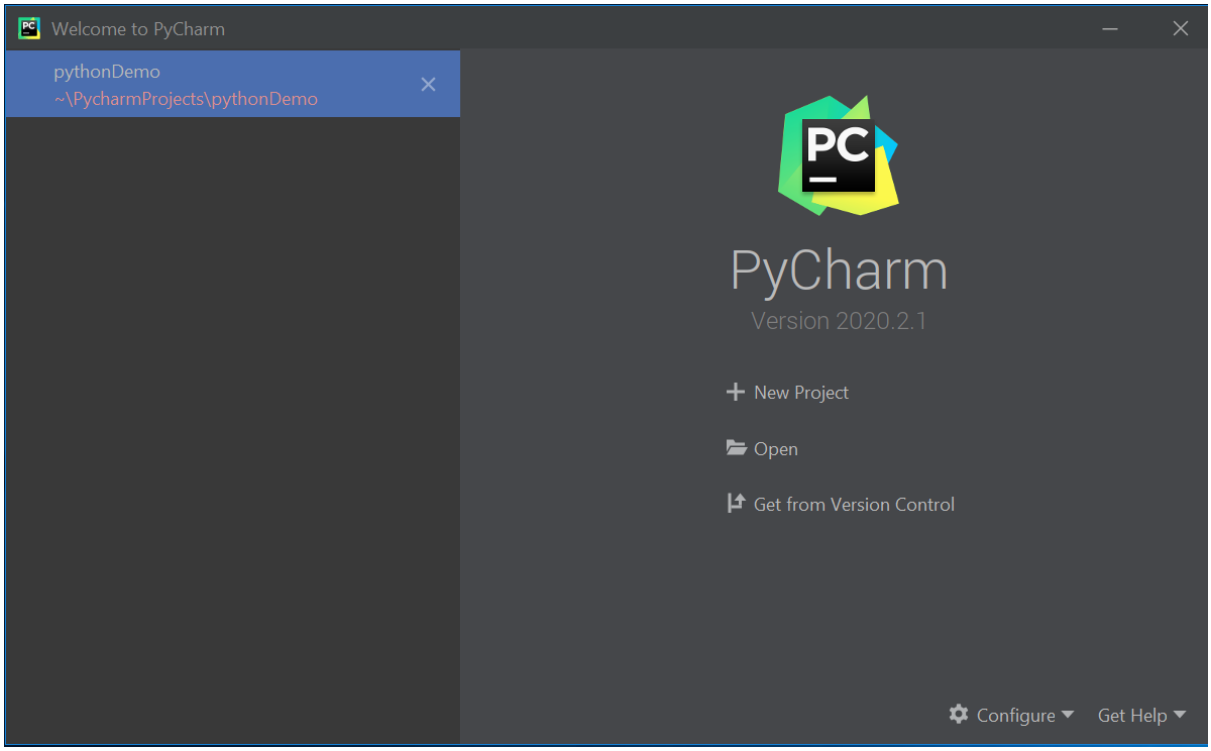
Introduction

In this workshop, we will work with PyCharm IDE through the DataFrame Syntax. PyCharm, created by the Czech company JetBrains, is a popular Integrated Development Environment (IDE) used in programming, particularly for the Python programming language. PyCharm works with Windows, macOS, and Linux versions. If you have experience with pandas in Python and SQL, you will find the structure of DataFrame is very similar with them.

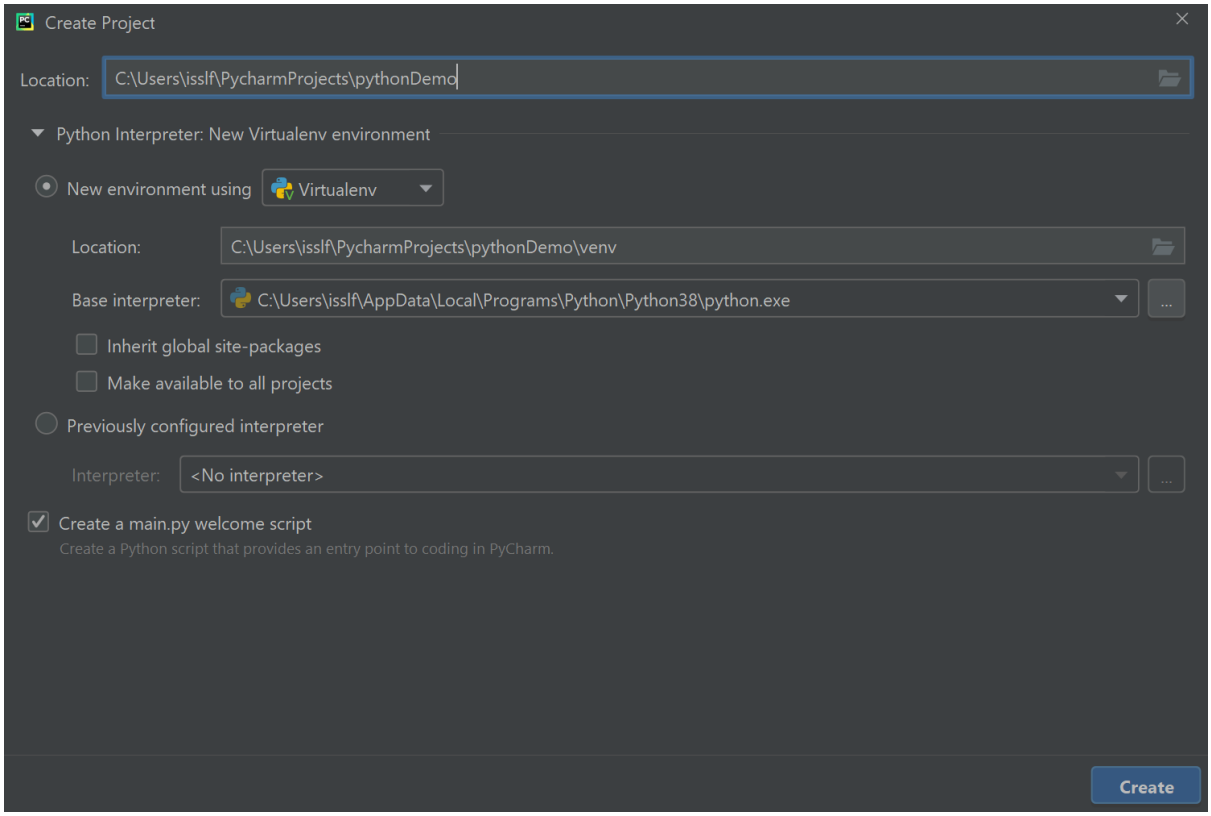
A DataFrame is a Dataset organized into named columns. Spark DataFrames hold data in a column and row format. Each column represents some feature or variable. Each row represents an individual data point. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs. We can then use these DataFrames to apply various transformations on the data. At the end of the transformation calls, we can either show or collect the results to display or for some final processing. In this workshop, we will cover the main features of DataFrame.

Create Python Project in PyCharm

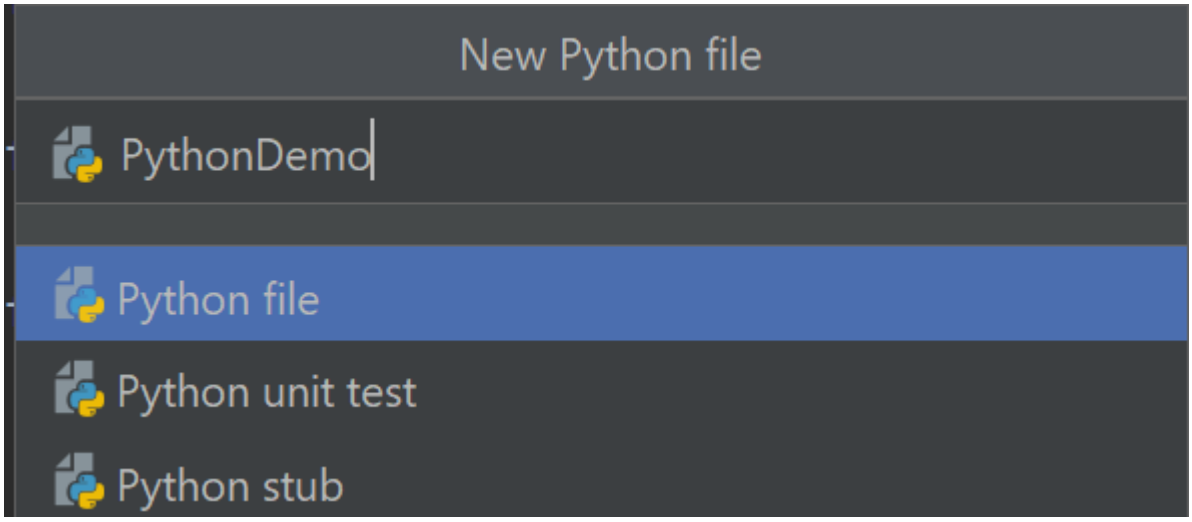
1. Click “New Project” in the File menu:



2. Give a meaningful project name, such as “PythonDemo”:



3. Now that we have created a Python project named as “PythonDemo”, it’s time to create a Python program file to write and run our first Python program. To create a file, right click on the folder name > New > Python file (as shown in the screenshot). Give the file name as “PythonDemo”:



4. We will use this script to help you to understand Python Basics. Please note this is not a comprehensive overview of Python programming. The purpose is to help you to have basic programing skills in Python and get prepare for the Machine Learning Workshop.

Data Types

In general, Python has the following built-in data types by default

- Data types
  - Numbers
  - Strings
  - Printing
  - Lists
  - Dictionaries
  - Booleans
  - Tuples

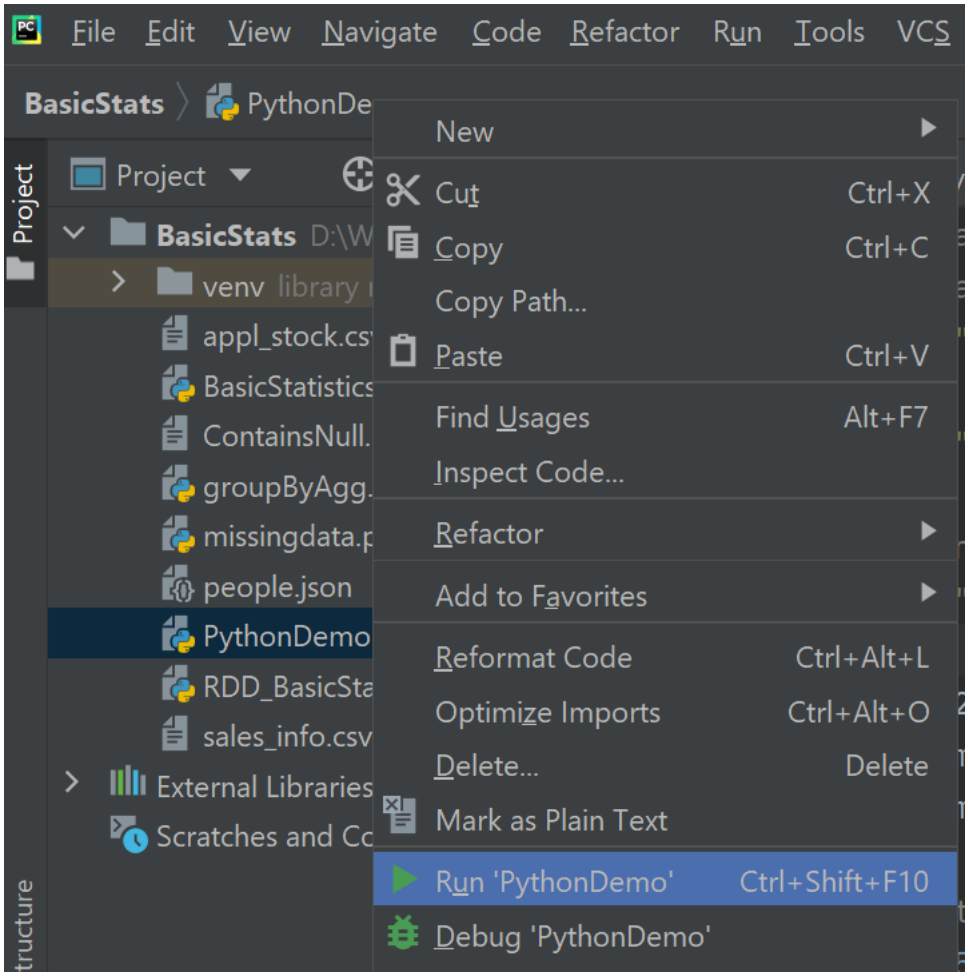
You can get the data type of any object by using the `type()` function:

```
# Variable Assignment
# variable can not start with number or special characters
print("Variable Assignment: ")
x = 2
print("x is {} ".format(x))


# String
print("String:")
my_str = 'my string'
my_str2 = "what's this?"
print(my_str)
print(my_str2)


# printing
num = 5
name = "Fan"
```

```
print('My number is: {one}, and my name is {two}'.format(one=num,two=name))
print('My number is: {}, and my name is {}'.format(num,name))


# Lists
my_list = ['a','b','c','d']
my_list.append('e')
print(my_list)
my_list[0] = 'new'
print(my_list)
nest_list = [1,2,3,[3,5,['target']]]
print(nest_list[3][2][0])


# Dictionaries
my_dic = {'k1':'val1','k2':'val2'}
print(my_dic['k1'])


# Tuple, tuple is immutable
my_tup = (1,2,3,4)
```

How to run Python script in PyCharm

Let's run the code. Right click on the PythonDemo.py file in the left sidebar and click on 'Run PythonDemo'.



You can see the outputs at the bottom of the screen.

Sample outputs are:

Variable Assignment:

```
x is 2
String:
my string
what's this?
My number is: 5, and my name is Fan
My number is: 5, and my name is Fan
['a', 'b', 'c', 'd', 'e']
['new', 'b', 'c', 'd', 'e']
target
val1
```

Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

if,elif, else Statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

```
# if statements
if 1 < 2:
    print('correct')
if 1 < 2:
    print('correct')
else:
```

```
    print('wrong')

if 1 == 2:
    print("first")
elif 3 == 3:
    print("middle")
else:
    print('last')
```

This snippet of codes will print different output based on different conditions.

For loop

For loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
# for loops
seq = [1,2,3,4,5]
for item in seq:
    print(item)
```

This snippet of code will print 1, 2, 3, 4, 5 to the console.

While loop

With the while loop we can execute a set of statements as long as a condition is true.

```
# While loops
i = 1
while i< 5:
    print('i is :{}'.format(i))
    i = i+1
```

Outputs are:

```
i is :1
i is :2
i is :3
i is :4
```

Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

In Python a function is defined using the `def` keyword:

```
# functions
def my_func(param1 = 'default'):
    """
    Dosstring goes here
    """
```

```
    print(param1)

my_func()
my_func('new param')
```

In this snippet of code, we define a function named as “my_func”, this function has one parameter “param1”, and the default value is ‘default’.

Lambda expression

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

```
lambda arguments : expression
```

Lambda expression example

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Output is:

```
13
```

Methods

Python has a set of built-in methods that you can use on string, lists/arrays, dictionary, tuple etc. The detailed description of built-in methods can refer to Python online documentation:

https://docs.python.org/3.7/

This example shows some common used methods on string.

```
# methods
my_string = 'Welcome to PyCharm course'
print(my_string.lower())
print(my_string.upper())
print(my_string.split())
```

Outputs are:

```
welcome to pycharm course
WELCOME TO PYCHARM COURSE
['Welcome', 'to', 'PyCharm', 'course']
```

Spark DataFrame Basics

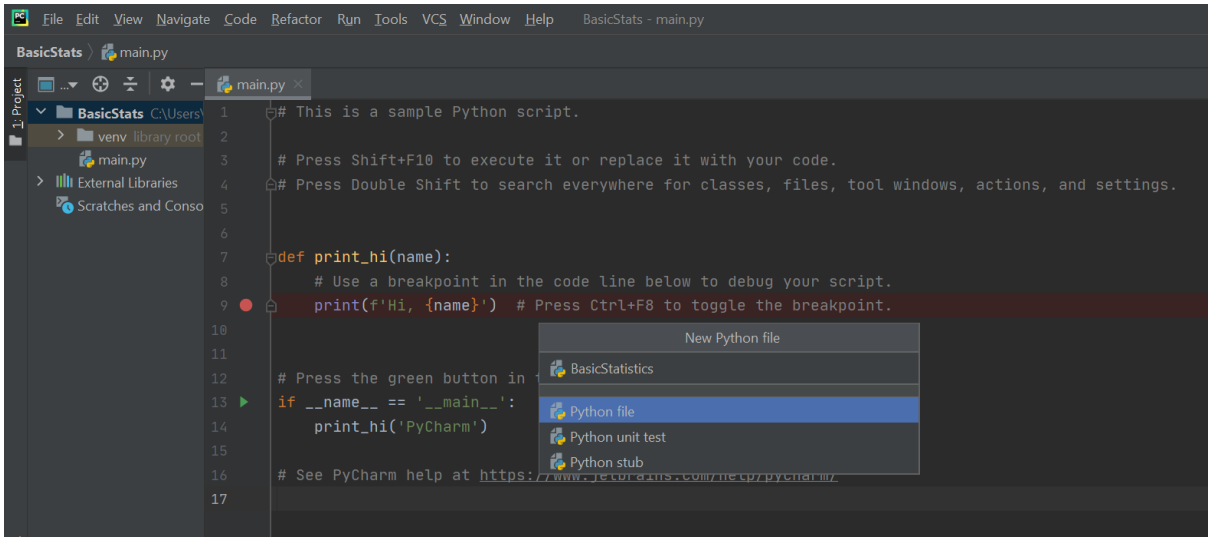
In this session, we will cover basic operations on DataFrame. Such as how to create a DataFrame, show data, retrieve data and create new columns.

Example 1

The data file we are using in this example is “people.json”. There are two attributes: name and age.

We will create a new Project named as “BasicStats”. Go to File -> New Project, name the new Project as “BasicStats”

Create a new python file name as “BasicStatistics.py”:




```
from pyspark.sql import SparkSession
# Start the SparkSession
spark = SparkSession.builder.appName("Basics Stats").getOrCreate()
# Read people.json file
df = spark.read.json('people.json')
# showing data
df.show()
# print schema
df.printSchema()
df.columns
df.describe()
# Retrieve data
df['age']
df.select('age').show()
# Returns list of Row objects
df.head(2)
# Return multiple columns
df.select(['age','name']).show()
```

In this example, we create a SparkSession by using the following builder pattern:

spark = SparkSession.builder.appName("Basics Stats").getOrCreate()

builder

A class attribute having a Builder to construct SparkSession instances.

appName(*name*)

Sets a name for the application, which will be shown in the Spark web UI.

If no application name is set, a randomly generated name will be used

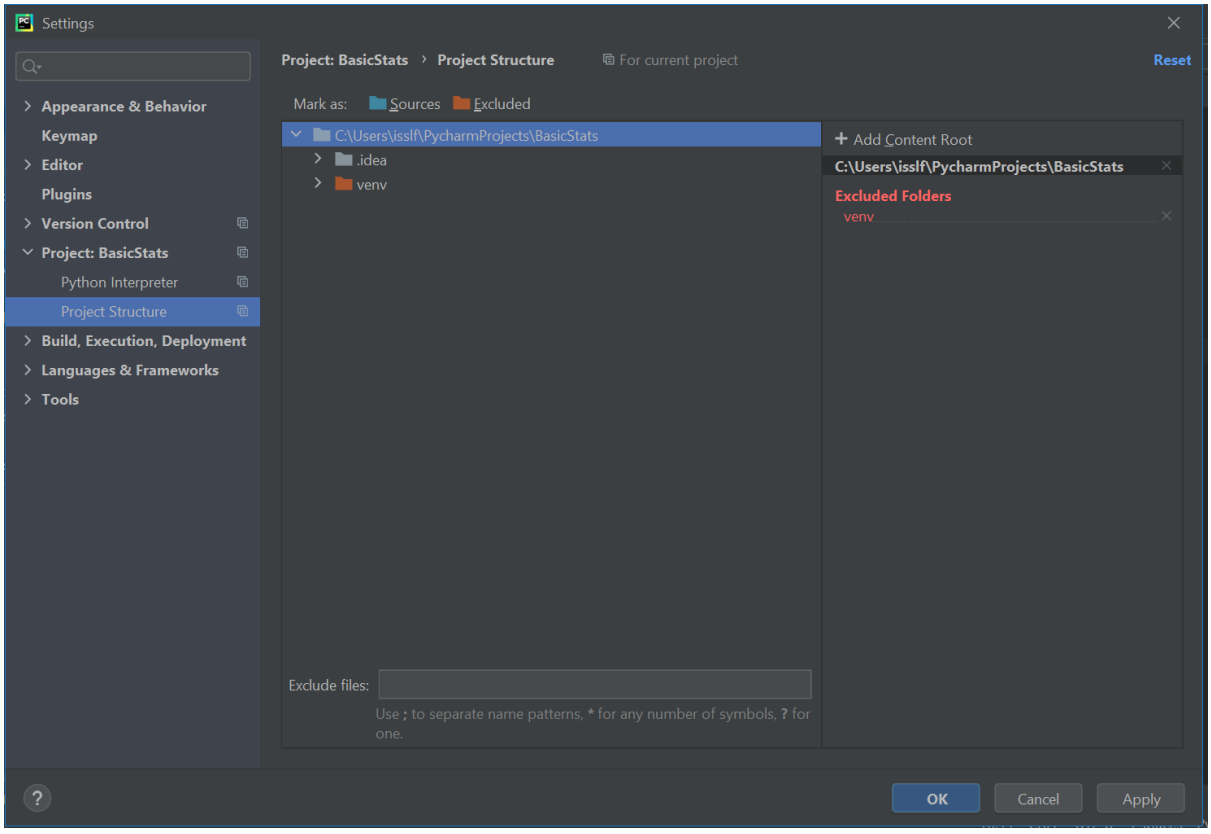
`getOrCreate()`

Gets an existing SparkSession or, if there is no existing one, creates a new one based on the options set in this builder.

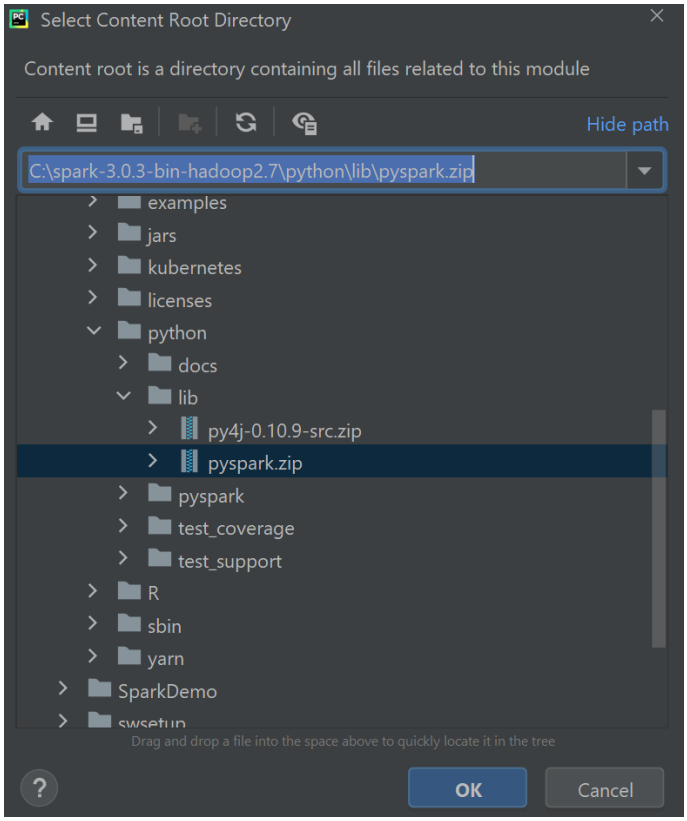
The detailed description can be found here:
https://spark.apache.org/docs/latest/api/python/pyspark.sql.html

In order to run PySpark in PyCharm, you need to "add Content Root", where you specify the location of the python executable of apache-spark.

- Go to File -> Setting -> navigate to Project Structure -> Click on 'Add Content Root'



- Go to folder where Spark is setup -> python -> lib -> py4j-0.10.9-src.zip and pyspark.zip and apply the changes and wait for the indexing to be done



- Press “Ok” and “Apply” after you are done. Return to Project window.

Sample outputs are as follows:

`df.show():`

```
+----+--------+
| age|    name|
+----+--------+
|null|  Martin|
|  30|    Andy|
|  19|  Justin|
|  20|    Jane|
|  23|Samantha|
+----+--------+
```

`df.printSchema():`

```
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

`df.select('age').show():`

```
+----+
| age|
+----+
|null|
|  30|
|  19|
|  20|
|  23|
+----+
```

`df.select(['age','name']).show():`

```
+----+--------+
| age|    name|
+----+--------+
|null|  Martin|
|  30|    Andy|
|  19|  Justin|
|  20|    Jane|
|  23|Samantha|
+----+--------+
```

Create new columns

```
# Adding a new column with a simple copy
df.withColumn('newage',df['age']).show()
df.withColumnRenamed('name','given_name').show()
df.withColumn('doubleage',df['age']*2).show()
```

This snippet of codes show how to create new columns.

Outputs are as follows:

```
+----+--------+------+
| age|    name|newage|
+----+--------+------+
|null|  Martin|  null|
|  30|    Andy|    30|
|  19|  Justin|    19|
|  20|    Jane|    20|
|  23|Samantha|    23|
+----+--------+------+

+----+----------+
| age|given_name|
+----+----------+
|null|    Martin|
|  30|      Andy|
|  19|    Justin|
|  20|      Jane|
|  23|  Samantha|
+----+----------+

+----+--------+---------+
| age|    name|doubleage|
+----+--------+---------+
|null|  Martin|     null|
|  30|    Andy|       60|
|  19|  Justin|       38|
|  20|    Jane|       40|
|  23|Samantha|       46|
+----+--------+---------+
```

Groupby and Aggregate Operations

Similar to SQL GROUP BY clause, PySpark `groupBy()` function is used to collect the identical data into groups on DataFrame and perform aggregate functions on the grouped data. In this session, we will discuss how to use groupBy and Aggregate methods on a DataFrame. groupBy allows you to group

rows together based off some column value, for example, you could group together sales data by the day the sale occurred, or group repeats customer data based off the name of the customer. Once you've performed the groupBy operation you can use an aggregate function off that data. An aggregate function aggregates multiple rows of data into a single output, such as taking the sum of inputs, or counting the number of inputs.

When we perform groupBy() on PySpark Dataframe, it returns GroupedData object which contains below aggregate functions.

`count()` - Returns the count of rows for each group.

`mean()` - Returns the mean of values for each group.

`max()` - Returns the maximum of values for each group.

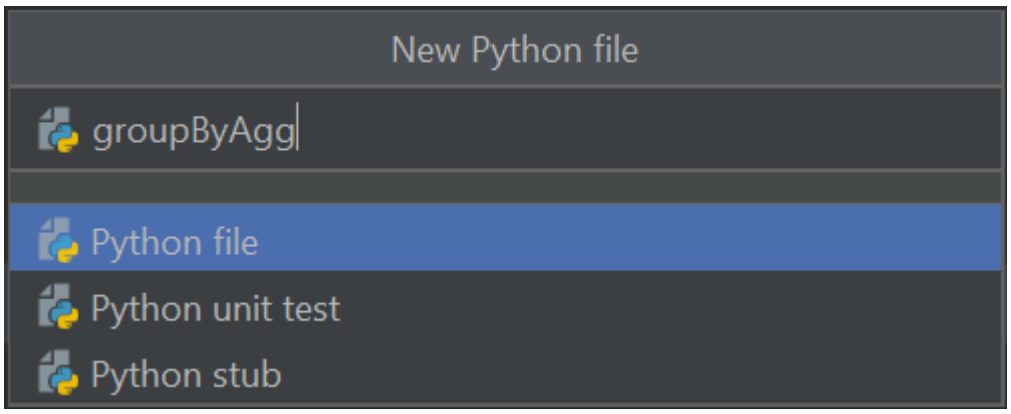
`min()` - Returns the minimum of values for each group.

`sum()` - Returns the total for values for each group.

`avg()` - Returns the average for values for each group.

Example 2

We will use the sales data as our dataset, named as sales_info.csv. It has three columns: Company, Person, Sales. Create a new python file name as “groupByAgg.py”:




```
from pyspark.sql import SparkSession
# Start Spark Session
spark = SparkSession.builder.appName("groupbyagg").getOrCreate()
# Read sales data
df = spark.read.csv('sales_info.csv',inferSchema=True,header=True)
df.printSchema()
df.show()
# Group by company
df.groupBy("Company")
# Returns a GroupedData object, off of which you can all various methods
# Mean
df.groupBy("Company").mean().show()
```

```
# Count
df.groupBy("Company").count().show()
# Max
df.groupBy("Company").max().show()
# Min
df.groupBy("Company").min().show()
# Order by sales ascending
df.orderBy("Sales").show()
# Descending call off the column itself.
df.orderBy(df["Sales"].desc()).show()
```

Outputs are as follows:

df.printSchema():

```
root
 |-- Company: string (nullable = true)
 |-- Person: string (nullable = true)
 |-- Sales: double (nullable = true)
```

df.show():

```
+-------+-------+-----+
|Company| Person|Sales|
+-------+-------+-----+
|   GOOG|    Sam|200.0|
|   GOOG|Charlie|120.0|
|   GOOG|  Frank|340.0|
|   MSFT|   Tina|600.0|
|   MSFT|    Amy|124.0|
|   MSFT|Vanessa|243.0|
|     FB|   Carl|870.0|
|     FB|  Sarah|350.0|
|   APPL|   John|250.0|
|   APPL|  Linda|130.0|
|   APPL|   Mike|750.0|
|   APPL|  Chris|350.0|
+-------+-------+-----+
```

df.groupBy("Company").mean().show():

```
+-------+-----------------+
|Company|       avg(Sales)|
+-------+-----------------+
```

```
|   APPL|            370.0|
|   GOOG|            220.0|
|     FB|            610.0|
|   MSFT|322.3333333333333|
+-------+-----------------+
```

df.groupBy("Company").count().show():

```
+-------+-----+
|Company|count|
+-------+-----+
|   APPL|    4|
|   GOOG|    3|
|     FB|    2|
|   MSFT|    3|
+-------+-----+
```

df.groupBy("Company").max().show():

```
+-------+----------+
|Company|max(Sales)|
+-------+----------+
|   APPL|     750.0|
|   GOOG|     340.0|
|     FB|     870.0|
|   MSFT|     600.0|
+-------+----------+
```

df.groupBy("Company").min().show():

```
+-------+----------+
|Company|min(Sales)|
+-------+----------+
|   APPL|     130.0|
|   GOOG|     120.0|
|     FB|     350.0|
|   MSFT|     124.0|
+-------+----------+
```

df.orderBy("Sales").show():

```
+-------+-------+-----+
|   GOOG|Charlie|120.0|
|   MSFT|    Amy|124.0|
|   APPL|  Linda|130.0|
```

```
|   GOOG|    Sam|200.0|
|   MSFT|Vanessa|243.0|
|   APPL|   John|250.0|
|   GOOG|  Frank|340.0|
|     FB|  Sarah|350.0|
|   APPL|  Chris|350.0|
|   MSFT|   Tina|600.0|
|   APPL|   Mike|750.0|
|     FB|   Carl|870.0|
+-------+-------+-----+
df.orderBy(df["Sales"].desc()).show()
+-------+-------+-----+
|Company| Person|Sales|
+-------+-------+-----+
|     FB|   Carl|870.0|
|   APPL|   Mike|750.0|
|   MSFT|   Tina|600.0|
|     FB|  Sarah|350.0|
|   APPL|  Chris|350.0|
|   GOOG|  Frank|340.0|
|   APPL|   John|250.0|
|   MSFT|Vanessa|243.0|
|   GOOG|    Sam|200.0|
|   APPL|  Linda|130.0|
|   MSFT|    Amy|124.0|
|   GOOG|Charlie|120.0|
+-------+-------+-----+
```

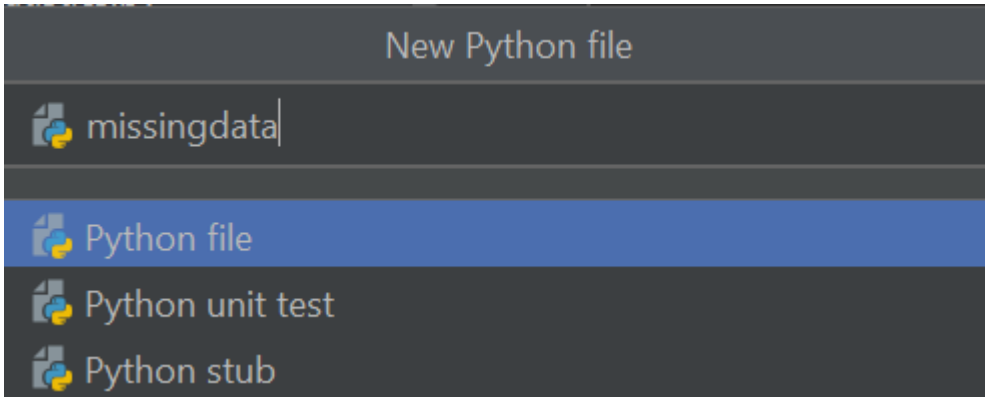
Missing Data

Data sources sometimes include missing data. There three basic methods to handle missing data

- Keep the missing data
- Remove missing data
- Fill them in with other value

Example 3

In this example, we will learn all three methods to handle missing data. The dataset is "ContainsNull.csv". It has three columns: Id, Name, Sales. Column Name and Sales contain missing data. Create a new python file name as "missingdata.py":



Keep missing data points

```
from pyspark.sql import SparkSession
# Start SparkSession
spark = SparkSession.builder.appName("missingdata").getOrCreate()
df = spark.read.csv("ContainsNull.csv",header=True,inferSchema=True)
df.show()
```

Remove missing data point

You can use the .na functions for missing data. The detailed description can be found here: https://spark.apache.org/docs/2.3.0/api/python/pyspark.sql.html#pyspark.sql.DataFrameNaFunctions.

```
# Drop any row that contains missing data
df.na.drop().show()
# Has to have at least 2 NON-null values
df.na.drop(thresh=2).show()
# Drop row “Sales” contains missing data
df.na.drop(subset=["Sales"]).show()
# Drop any row contains missing data
df.na.drop(how='any').show()
# Drop those rows contains missing data for all columns
df.na.drop(how='all').show()
```

Outputs are as follows:

`df.show()`:

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John| null|
|emp2| null| null|
|emp3| null|345.0|
```

```
|emp4|Cindy|456.0|
+----+-----+-----+
```

df.na.drop().show():

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp4|Cindy|456.0|
+----+-----+-----+
```

df.na.drop(thresh=2).show():

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John| null|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

df.na.drop(subset=["Sales"]).show():

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

df.na.drop(how='any').show():

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp4|Cindy|456.0|
+----+-----+-----+
```

df.na.drop(how='all').show():

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John| null|
|emp2| null| null|
|emp3| null|345.0|
|emp4|Cindy|456.0|
```

```
+----+-----+-----+
```

Fill missing data point

We can also fill the missing values with new values. If you have multiple nulls across multiple data types, Spark is actually smart enough to match up the data types.

```
# Fill missing data with “NEW VALUE” for string data type only
df.na.fill('NEW VALUE').show()
# Fill missing data with 0 for numeric data type only
df.na.fill(0).show()
# Fill missing data in row “Name” with “No Name”
df.na.fill('No Name',subset=['Name']).show()
# Fill values with mean value for column “Sales”
import pyspark.sql.functions as F
mean_val = df.select(F.mean(df['Sales'])).collect()
# Weird nested formatting of Row object!
mean_val[0][0]
mean_sales = mean_val[0][0]
df.na.fill(mean_sales,["Sales"]).show()
```

Outputs are as follows:

```
df.na.fill('NEW VALUE').show()
+----+---------+-----+
|  Id|     Name|Sales|
+----+---------+-----+
|emp1|     John| null|
|emp2|NEW VALUE| null|
|emp3|NEW VALUE|345.0|
|emp4|    Cindy|456.0|
+----+---------+-----+
df.na.fill(0).show()
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John|  0.0|
|emp2| null|  0.0|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
df.na.fill('No Name',subset=['Name']).show()
+----+-------+-----+
```

```
|  Id|   Name|Sales|
+----+-------+-----+
|emp1|   John| null|
|emp2|No Name| null|
|emp3|No Name|345.0|
|emp4|  Cindy|456.0|
+----+-------+-----+
df.na.fill(mean_sales,["Sales"]).show()
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John|400.5|
|emp2| null|400.5|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

Summary

In this workshop, we have discussed the basics of Spark DataFrame, including basic operations, groupBy and Aggregation operations, as well as how to handle missing data using PySpark.

After we have solid understanding of Spark DataFrame, we can move on to utilizing the DataFrame MLlib API for Machine Learning.

Exercise

You are supposed to create PyCharm Project and write Python codes to answer the following Questions.

Q1 This question is test your understanding of Python Basics.

a) What is 8 to the power 4?
b) Split this string “Split this string”
c) Given the variables: planet = “Earth”, diameter = 12742, use .format() to print the following string “The diameter of Earth is 12742 kilometers.”
d) Given the name list, use indexing to grab word “target”, the_list = [1,2,[3,4],[5,[100,200,['target']],23,11],1,7]
e) Given this nest dictionary grab the work “hello”. The_dic = {'k1':[1,2,3,{'tricky':['oh','man','inception',{'target':[1,2,3,'hello']}]}]}
f) Create a basic function that returns True if the word 'elephant' is contained in the input string. Don't worry about edge cases like a punctuation being attached to the word dog, but do account for capitalization.
g) Create a function that counts the number of times the word "elephant" occurs in a string. Again ignore edge cases.

h) Write a function to return one of 3 possible results: "Low speed", "Medium speed", or "Fast speed". If your speed is 60 or less, the result is "Low speed". If speed is between 61 and 80 inclusive, the result is "Medium speed". If speed is 81 or more, the result is "Fast speed". Unless it is your birthday (encoded as a boolean value in the parameters of the function) -- on your birthday, your speed can be 5 higher in all cases.

Q2 This question is test your understanding of PySpark operations on DataFrame (Optional).

Given the following DataFrame which contains "**employee_name**", "**department**", "**country**", "**salary**", "**age**" and "**bonus**" columns

```
Data = [("James","Sales","SG",70000,34,10000),
        ("Michael","Sales","SG",66000,56,20000),
        ("Robert","Sales","MY",61000,30,23000),
        ("Maria","Finance","MY",60000,24,23000),
        ("Raman","Finance","USA",79000,40,24000),
        ("Scott","Finance","USA",63000,36,19000),
        ("Jen","Finance","UK",89000,53,15000),
        ("Jeff","Marketing","UK",70000,25,18000),
        ("Alice","Marketing","UK",78000,50,21000),
        ("Ada","IT","SG",83000,35,11000),
        ("Jackson","IT","MY",71000,30,21000),
        ("Cooper","IT","UK",91000,40,21000)]
```

Complete the following items:

a) Create a PySpark DataFrame based on the given RDD.
b) Show data and print schema
c) Run groupBy() on "department" columns. Calculate aggregates like minimum, maximum, average, total salary for each group using min(), max(), avg() and sum() aggregate functions respectively.
d) Run groupBy() on "country" columns. Calculate aggregates like minimum, maximum, average, total salary for each group using min(), max(), avg() and sum() aggregate functions respectively.

Workshop submission

Please complete all the exercise questions, zip all the files and scripts, name as *workshop10_your name.zip* and upload into canvas respective submission folder.