6.6. Python编程实践

▲6.5.大数据分析技术与Python

▼6.7. 继续学习本章知识

Python 编程实践

【分析对象】

txt 文件—文件名为“Pearson.txt”，该数据源自卡尔·皮尔森（Karl Pearson）的著名实验，主要记录的是父亲和儿子的身高。该数据集在 1 078个样本数据的基础上，增加了随机噪声。读者可以从Kaggle官网下载或在本书配套资源中找到数据文件“Pearson.txt”。

【分析目的与任务】

理解 Spark+ MongoDB 在数据科学中的应用—进行大数据分析。

首先，创建 SparkSession，并连接至 Spark 服务器和 MongoDB 服务器 其次，通过查看模式信息、描述性统计信息和进行数据可视化来理解数据接着，处理离群点，并将父亲的身高作为自变量，取他的成年儿子的身高作为因变量进行简单线性回归最后，评价拟合出来的模型，利用模型进行预测.

【分析方法及工具】

Python 及 Spark+ MongoDB 大数据分析框架。

Python 编程实践

【主要步骤】

- 此数据科学项目的**步骤**包括创建 SparkSession、数据读入、数据理解、数据准备、数据准备、模型训练、模型评估和模型应用。
- 本例以简单线性回归为例，讲解基于 Spark 和 MongoDB 的大数据分析思路和方法。与本书“2.7Pyhton 编程实践”中介绍的基于 statsmodels 包的简单线性回归不同的是，本例将采用 Spark 的机器学习库—**MLib 作为简单线性回归**的工具。
- 在编写本例题代码之前，读者需要在自己的机器上安装 Spark 编程环境、PySpark 包和 MongoDB 服务器。由于篇幅限制，本书中并没有讲解上述内容的具体操作方法， 读者都可以在本书配套资源中找到相关知识的介绍文档。
- 在 Spark 编程中，首先需要创建 SparkSession 对象，该对象用于将 Python 会话连接至 Spark 服务器和数据库服务器（如 MongoDB 等）中。创建 SparkSession 对象的方 法为调用 pyspark.sql 包提供的 **SparkSession()函数**。该函数的调用方法如下。

Step1：创建 SparkSession

```
In[1]|#创建 SparkSession，并连接至 Spark 服务器和 MongoDB 服务器
     |from pyspark.sql import SparkSession
     |mySpark = SparkSession \
     |.builder \
     |.appName("myApp") \
     |.config("spark.mongodb.input.uri", "mongodb://127.0.0.1/local.
     |FSHeight") \
     |.config("spark.mongodb.output.uri", "mongodb://127.0.0.1/local.
     |FSHeight") \
     |.config('spark.jars.packages','org.mongodb.spark:mongo-spark
     |-connector_2.11:2.4.1‘)\
     |.getOrCreate()
```

上传 MongoDB

在数据科学项目中，待分析处理的原始数据集可能已经存放在 MongoDB 等数据库服务器中，也可能存放在本地文件系统中。例如，本例中的原始数据集“Pearson.txt”是存放在本地文件系统中，需要将其上传至 MongoDB。在 Python 编程中，通常采用 PySpark 包将本地数据文件上传 MongoDB，具体方法为：调用 PySpark 的 read()方法将数据文件 FSHeight.txt 中的数据读入至自定义的 Spark 数据框 sparkFSHeight 中。示例如下。

In[2]
```
#用 PySpark 的 read()方法将数据文件 FSHeight.txt 中的数据读入至自定义的 Spark 数据框 sparkFSHeight 中
sparkFSHeight=mySpark.read.format("csv")\
.option("inferSchema","true")\
.option("delimiter",'\t')\
.option("header", "true")\
 .load("FSHeight.txt")
```

显示 Spark 数据框 sparkFSHeight 的前五行

其次，可以通过调用 PySpark 提供的 show()函数显示 Spark 数据框 sparkFSHeight 的部分内容。示例如下。

```
In[3]: #显示 Spark 数据框 sparkFSHeight 的前五行
       sparkFSHeight.show(5)
```

对应输出结果为：

```
+------+----+
|Father| Son|
+------+----+
|  65.0|59.8|
|  63.3|63.2|
|  65.0|63.3|
|  65.8|62.8|
|  61.1|64.3|
+------+----+
only showing top 5 rows
```

显示Spark数据框 sparkFSHeight 的行数、模式信息

接着，可以通过 PySpark包提供的其他函数，如 count()、printSchema()等实现查看数据集的行数和模式信息等数据理解的目的。

```
In[4]|#显示 Spark 数据框 sparkFSHeight 的行数
     | sparkFSHeight.count()
```

对应输出结果为：

```
1078
```

从以上数据结果可以看出，原始数据 FSHeigh 的行数为 1078。随后，查看其模式信息。示例如下。

```
In[5]|#显示 Spark 数据框 sparkFSHeight 的模式信息
     | sparkFSHeight.printSchema()
```

对应输出结果为：

```
root
|-- Father: double (nullable = true)
|-- Son: double (nullable = true)
```

通过 PySpark 包将 Spark 数据框上传至 MongoDB 服务中

再次，我们通过 PySpark 包将 Spark 数据框上传至 MongoDB 服务中。具体方法为：以“追加”方式将 Spark 数据框 sparkFSHeight 存入 MongoDB 的 local 数据库的数据 集 FSHeight 中。示例如下。

```
In[6] sparkFSHeight.write.format(“mongo”).option(“uri”,“mongodb://127.
      0.0.1/local.FSHeight").mode("append").save()
      #【注意】：本例是采用  append()形式追加新数据的，所以，多次运行此行则数据重复写入。
      monggodb 中清空数据表的命令为 db.FSHeight.remove({})
```

最后，调用 stop()方法，关闭 Spark Session 对象。示例如下。

```
In[7] mySpark.stop()
```

Step 2:数据读入

在数据科学流程上看，与本书“2.7Pyhton 编程实践”中介绍过的基于统计学方法进行数据建模方法类似，基于 Spark 的大数据分析也需要进行业务理解、数据读入和数据理解等活动。

PySpark 中可以通过创建 SparkSession，并调用其 read.format()函数从 MongoDB 数据库中读取数据。在本例的Step1中，已经将数据集写入MongoDB的local数据库的FSHeight 数据集中。因此，从MongoDB 读取数据集FSHeight 的代码如下。

```
In[8] #创建 SparkSession 对象
      from pyspark.sql import SparkSession
      mySpark = SparkSession \
      .builder \
      .appName("myApp") \
      .config("spark.mongodb.input.uri", "mongodb://127.0.0.1/local.FSHeight") \
      .config("spark.mongodb.output.uri", "mongodb://127.0.0.1/local.FSHeight") \
      .config('spark.jars.packages','org.mongodb.spark:mongo  spark connector_2.11:2.4.1')\
      .getOrCreate()
```

其次，从 MongoDB 读取数据集 FSHeight 至 Spark 数据框中。示例如下。

```
In[9]sparkDF_FSHeight
     sparkDF_FSHeight=mySpark.read.format("mongo").load()
```

再次，显示 Spark 数据框 sparkDF_FSHeight 的前 5 行。示例如下。

```
In[10]sparkDF_FSHeight.head(5)
      #【提示】此处，可以调用 take()方法实现 head()方法的功能
```

对应输出结果

```
[Row(Father=65.0, Son=59.8, _id=Row(oid='5e0958f52c796e5c28a79a01')),
Row(Father=63.3, Son=63.2, _id=Row(oid='5e0958f52c796e5c28a79a02')),
Row(Father=65.0, Son=63.3, _id=Row(oid='5e0958f52c796e5c28a79a03')),
Row(Father=65.8, Son=62.8, _id=Row(oid='5e0958f52c796e5c28a79a04')),
Row(Father=61.1, Son=64.3, _id=Row(oid='5e0958f52c796e5c28a79a05'))]
```

最后，查看 Spark 数据框 sparkDF_FSHeight 的行数。示例如下。

```
In[11] sparkDF_FSHeight.count()
       #【提示】如果显示行数并非为 1078，那么说明之前多次插入数据集或数据集已
       发生变化，建议清空 collections 后执行以上代码，清空 collections 的具体代码为：
       db.FSHeight.remove({})
```

对应输出结果为：

```
1078
```

Step 3:数据理解

基于Spark 数据理解的思路与本书第2章和第3章中讲解的基于统计学或机器学习的数据理解类似，其区别在于所调用的函数不同。

在PySpark中，可以调用 printSchema()、describe()和 take()等方式实现查看数据框的模式信息、述性统计信息和部分数据内容等功能。当然，也可以采用数据可视化的方法实现数据理解的目的。

（1）查看 Spark 数据框—sparkDF_FSHeight 的模式信息

首先，查看 Spark 数据框—sparkDF_FSHeight 的模式信息。示例如下。

```
In[12]|sparkDF_FSHeight.printSchema()
```

对应输出结果为：

```
root
|-- Father: double (nullable = true)
|-- Son: double (nullable = true)
|-- _id: struct (nullable = true)
|              |-- oid: string (nullable = true)
```

（2）查看 Spark 数据框—sparkDF_FSHeight 的模式信息

其次，查看 Spark 数据框—sparkDF_FSHeight 的描述性统计信息。示例如下。

```
In[13] sparkDF_FSHeight.describe().toPandas().transpose()
```

对应输出结果为：

	0	1	2	3	4
summary	count	mean	stddev	min	max
Father	1078	67.68682745825602	2.745827077877217	59.0	75.4
Son	1078	68.68423005565862	2.8161940362006628	58.5	78.4

（3） 显示 Spark 数据框—sparkDF_FSHeight 的前 10 行

再次，显示 Spark 数据框—sparkDF_FSHeight 的前 10 行。示例如下。

```
In[14]|sparkDF_FSHeight.take(10)
```

对应输出结果为：

```
[Row(Father=65.0, Son=59.8, _id=Row(oid='5e0958f52c796e5c28a79a01’)),
 Row(Father=63.3, Son=63.2, _id=Row(oid='5e0958f52c796e5c28a79a02')),
Row(Father=65.0, Son=63.3, _id=Row(oid='5e0958f52c796e5c28a79a03')),
Row(Father=65.8, Son=62.8, _id=Row(oid='5e0958f52c796e5c28a79a04')),
Row(Father=61.1, Son=64.3, _id=Row(oid='5e0958f52c796e5c28a79a05')),
Row(Father=63.0, Son=64.2, _id=Row(oid='5e0958f52c796e5c28a79a06')),
Row(Father=65.4, Son=64.1, _id=Row(oid='5e0958f52c796e5c28a79a07')),
Row(Father=64.7, Son=64.0, _id=Row(oid='5e0958f52c796e5c28a79a08')),
Row(Father=66.1, Son=64.6, _id=Row(oid='5e0958f52c796e5c28a79a09')),
Row(Father=67.0, Son=64.0, _id=Row(oid='5e0958f52c796e5c28a79a0a'))]
```

（4）绘制“儿子身高-父亲身高”散点图

最后输出结果为：

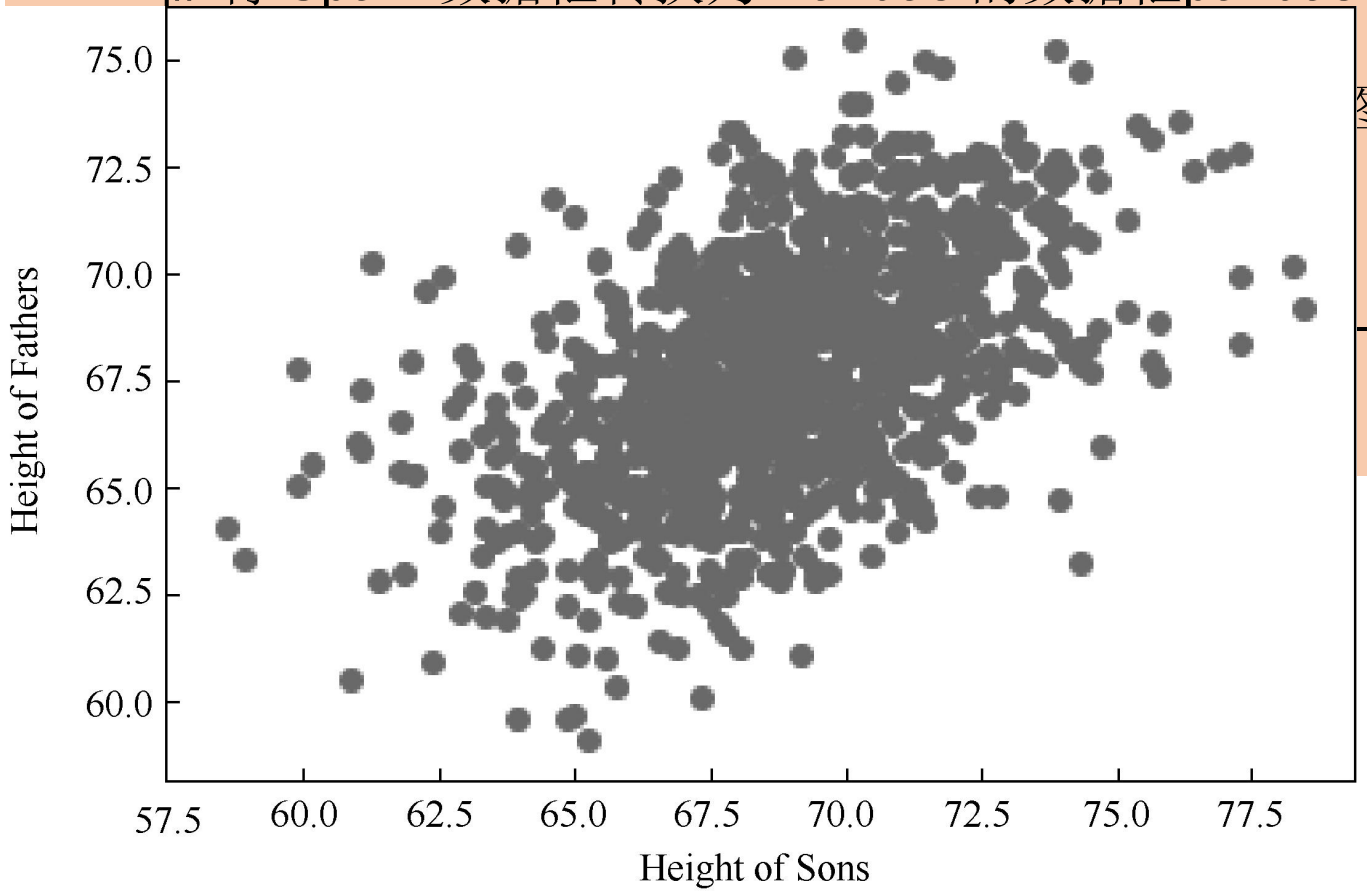
最后输出结果为“儿子身高-父亲身高”散点图。示例如下。

```
Text(0, 0.5, 'Height of Fathers')
```

```
# 将 Spark 数据框转换为 Pandas 的数据框pandasDF_FSHeight=sparkDF

图

_FSHeight["Father"])
```



Step 4:数据准备

在基于 Spark 的大数据分析中，除了离群点处理和缺失值处理等预处理活动外，主要是需要按照 MLib 包中对**数据模态**的要求进行数据准备。

（1）显示 Spark 数据框 pandasDF_FSHeight的部分内容

首先，调用 head()方法，显示 Spark 数据框 pandasDF_FSHeight的部分内容。示例如下。

```
In[16] | pandasDF_FSHeight.head()
```

对应输出结果为

	Father	Son	_id
0	65.0	59.8	(5e0958f52c796e5c28a79a01,)
1	63.3	63.2	(5e0958f52c796e5c28a79a02,)
2	65.0	63.3	(5e0958f52c796e5c28a79a03,)
3	65.8	62.8	(5e0958f52c796e5c28a79a04,)
4	61.1	64.3	(5e0958f52c796e5c28a79a05,)

（2）查看 Spark 数据框 pandasDF_FSHeight 的形状

其次，调用 shape 属性，查看 Spark 数据框 pandasDF_FSHeight 的形状。示例如下。

```
In[17]  pandasDF_FSHeight.shape
```

对应输出结果为

```
(1078, 3)
```

（3） 计算儿子身高和父亲身高的 z-score 值

接着，计算儿子身高和父亲身高的 z-score 值。示例如下。

```
In[18] from scipy import stats import numpy as np
       z = np.abs(stats.zscore(pandasDF_FSHeight[["Son","Father"]]))
       print(z)
```

对应输出结果为

```
[[3.1561581 0.97896716]
[1.94829456 1.59837581]
[1.91276917 0.97896716]
...
[0.21875472 1.49866744]
[0.21875472 1.09787361]
[0.59832943 0.9521304 ]]
```

（4） 删除离群点

接着，根据上一步中计算的 z-score 值，删除离群点。示例如下。

```
In[19]: zscored_pandasDF_FSHeight = pandasDF_FSHeight[(z < 3).all(axis=1)]
        #【注意】此处离群点转换后的数据集存放在对象
        该对象为 Pandas 包的数据框
        zscored_pandasDF_FSHeight，
        #查看 Pandas 数据框 zscored_pandasDF_FSHeight 的形状
        zscored_pandasDF_FSHeight.shape
```

对应输出结果为

```
(1067, 3)
```

（5）将离群点处理后的数据集 zscored_pandasDF_FSHeight 转换为 Spark 数据框 zscored_ sparkDF_FSHeight

从该输出结果可以看出，以 z-score 值为依据进行离群点的删除处理后，数据框 zscored_pandasDF_FSHeight 的行数变为 1067，即从数据框 pandasDF_FSHeight 删除了 共 11 行数据。由于数据框 zscored_pandasDF_FSHeight 和 pandasDF_FSHeigh 均为 Pandas 的数据框，接下来将离群点处理后的数据集 zscored_pandasDF_FSHeight 转换为 Spark 数据框 zscored_ sparkDF_FSHeight。示例如下。

```
In[20] zscored_sparkDF_FSHeight=mySpark.createDataFrame(zscored_panda sDF_FSHeight)
       #重新上传至 MongoDB 数据库服务器
       zscored_sparkDF_FSHeight.write.format(“mongo”).option(“uri”,“mongodb://127.0.0.1/
       local.Zscor     edFSHeight").mode("append").save()
       #从 MongoDB 数据库服务器重新读取数据文件
       sparkDF_FSHeight=mySpark.read.format("mongo").option("uri","mongodb://127.0.0.1/
       local.ZScoredFSHeight").load()
       #显示已读取的 Spark 数据框 sparkDF_FSHeigh 的行数和列数
        sparkDF_FSHeight.count(),len(sparkDF_FSHeight.columns)
```

对应输出结果为

```
(1067, 3)
```

（6）转换为特征矩阵和目标向量

与本书“2.7Pyhton 编程实践”中介绍过的基于统计学方法进行数据建模方法类似， 基于 Spark 的大数据分析也需要将自变量和因变量对应的数据分别转换为特征矩阵和目标向量。在 PySpark 中，通常采用 pyspark.ml.feature 包提供的 VectorAssembler()函 数，将多个列（用 inputCols=表示）合并成一个“向量列（vector column）（用 outputCol =表示），进而实现定义 MLlib 所需的特征矩阵的目的。示例如下。

```
In[21] #导入函数 VectorAssembler
       from pyspark.ml.feature import VectorAssembler
       #定义一个“向量列”（Vector Column）生成器
        vectorAssembler = VectorAssembler(inputCols = ['Father'], outputCol = 'features‘)
        #将多个列（用 inputCols=表示）合并成一个“向量列”（Vector Column）
        v_sparkDF_FSHeight = vectorAssembler.transform(sparkDF FSHeight)
        #显示 Spark 数据框 v_sparkDF_FSHeight 的部分内容 v_sparkDF_FSHeight.take(3)
```

对应输出结果为

```
[Row(Father=67.6, Son=68.2, _id=Row(oid='5e0958f52c796e5c28a79a5b'), features= DenseVector([67.6])),
Row(Father=68.4, Son=67.9, _id=Row(oid='5e0958f52c796e5c28a79a5c'), features= DenseVector([68.4])),
Row(Father=67.7, Son=68.6, _id=Row(oid='5e0958f52c796e5c28a79a5d'), features=DenseVector([67.7]))]
```

（7）显示数据框 v_sparkDF_FSHeight.的前3行数据

调用 select()方法从 Spark 数据框 v_sparkDF_FSHeight 中提取自变量“features”和因变量“Son”，存入另一个 Spark 数据框 v_sparkDF_FSHeight 中，并调用 take()方法显示数据框 v_sparkDF_FSHeight.的前 3 行数据 。

```
In[22]| v_sparkDF_FSHeight = v_sparkDF_FSHeight.select([‘features’, ‘Son‘])
      | v_sparkDF_FSHeight.take(3)
```

对应输出结果为

```
[Row(features=DenseVector([67.6]), Son=68.2),
 Row(features=DenseVector([68.4]), Son=67.9),
Row(features=DenseVector([67.7]), Son=68.6)]
```

（8）划分为测试集和训练集

值得一提的是，Spark 提供的 MLib 库采用机器学习方法实现简单线性回归，因此，需要将特征矩阵和目标向量进一步划分为测试集和训练集。关于测试集与训练集的划分方法，可以参见本书第 3 章的内容。示例如下。

In[23]
```
#训练集和测试集的划分
train_DF = v_sparkDF_FSHeight test_DF= v_sparkDF_FSHeight
#【提示】在此，考虑到本例是以简单回归为目的进行数据分析的，我们将训练集和测试集均设为整个数据集。通常，用.randomSplit 函数进行训练集和测试集的切分，代码如下
splits = v_sparkDF_FSHeight.randomSplit([0.7, 0.3])
train_DF = splits[0]
test_DF = splits[1]
#显示测试集的行数和列数
test_DF.count(),len(test_DF.columns)
```

对应输出结果为

```
(1067, 2)
```

Step 5:模型训练

在数据准备的基础上，Spark 可以调用MLib 中的具体函数进行各种机器学习的目的。本例调用Spark MLib 的LinearRegression 函数进行简单线性回归。
首先，导入Spark MLib 中的LinearRegression 函数，并创建和拟合LinearRegression模型。示例如下。

```
In[24]|#导入 LinearRegression 函数
      | from pyspark.ml.regression import LinearRegression
      |#创建 LinearRegression 模型
      | myModel = LinearRegression(featuresCol = 'features', labelCol= 'Son')
      |#【提示】featuresCol 和 labelCol 分别代表的是特征矩阵和目标向量
      |#拟合 LinearRegression 模型 myResults = myModel.fit(train_DF)
```

其中，关于 LinearRegression 函数的更多信息建议查看其帮助信息。帮助信息的查看方法如下。

```
In[25]|#查看 LinearRegression 函数的帮助信息
      | LinearRegression?
```

其次，通过调用 myResults 对象的 coefficients 和 intercept 属性，分别查看拟合得出的斜率和截距项。示例如下。

In[26]:
```
#显示斜率
print("Coefficients: " + str(myResults.coefficients))

#显示截距项
print("Intercept: " + str(myResults.intercept))
```

对应输出结果为：

```
Coefficients: [0.49182721146365943]
Intercept: 35.3942223308334
```

Step 6:模型评估

PySpark 为我们提供了可用于模型评估的属性和方法。例如，通过summary属性的 show()方法、r2 属性和.rootMeanSquaredError 属性等分别查看简单线性回归之后的残差、R方和均方误差。

（1）生成summary对象、查看残差项

首先，生成summary 对象。示例如下。

```
In[27] | summary = myResults.summary
```

其次，查看 summary 对象中的残差项。示例如下。

```
In[28] | #查看残差
       | summary.residuals.show()
```

对应输出结果为：

```
+-------------------+
|          residuals|
+-------------------+
|-0.44174182577677357|
| -1.1352035949477113|
|-0.09092454692314789|
| -1.1827517583868143|
| -1.1778480852649977|
| -1.2827517583868087|
| -0.779482642972269|
| -1.277848085264992|
|                -0.53356903724044|
| -1.525396248704098|
| -1.6188580178750271|
| -2.612319787045962|
| 2.1500853856868787|
| -1.6155889024605017|
| 0.8139791261986602|
| -2.7582333927777825|
| 1.1107100107841177|
| -3.6451569311196437|
| 1.4615272896377576|
| 1.7139791261986659|
+-------------------+
only showing top 20 rows
```

（2）查看判断系数 R 方、均方误差

再次，查看判断系数 R 方。示例如下。

```
In[29] |   summary.r2
```

对应输出结果为：

```
0.2502440487185261
```

最后，查看均方误差。示例如下。

```
In[30] | summary.rootMeanSquaredError
```

对应输出结果为：

```
2.325165606265826
```

Step 7:模型应用

在模型评估通过后，可以将已训练好的模型用于解决实际问题。本例中，我们调用拟合后生成的LinearRegression模型—myResults的transform()方法实现预测功能。

（1）显示预测结果

首先，采用所训练出的新模型myResults 对测试集中的儿子(Son)的身高进行预测，并对比显示预测结果与测试集中的真实值。示例如下。

```
In[31]|#基于测试集 test_DF，进行预测
      |predictions = myResults.transform(test_DF)
      |#显示预测结果 predictions.show()
```

对应输出结果为：

```
+--------+----+----------------+
|features| Son|      prediction|
+--------+----+----------------+
| [67.6]|68.2|68.641741825776 78|
| [68.4]|67.9|69.03520359494772|
| [67.7]|68.6|68.69092454692314|
| [68.7]|68.0|69.18275175838681|
| [69.3]|68.3| 69.477848085265|
| [68.7]|67.9|69.18275175838681|
| [69.1]|68.6|69.37948264297226|
| [69.3]|68.2| 69.477848085265|
|              [68.6]|68.6|69.13356903724043|
[69.6]|68.1|69.62539624870409|
[70.4]|68.4|70.01885801787503|
[71.2]|67.8|70.41231978704596|
[66.6]|70.3|68.14991461431312| [70.8]|68.6|
70.2155889024605| [68.3]|69.8|68.986020873801 34|
[71.7]|67.9|70.65823339277779|
[67.9]|69.9|68.78928998921589|
[73.3]|67.8|71.44515693111964|
[68.0]|70.3|68.83847271036224|
[68.3]|70.7|68.98602087380134|
+--------+----+----------------+
only showing top 20 rows
```

（2）显示为 prediction 的一列的部分内容

其次，调用 show()方法，从上一结果中选择名为 prediction 的一列，并显示其部分内容。示例如下 。

```
In[32] predictions.select("prediction").show()
```

对应输出结果为：

```
+----------------+
|              prediction|
+----------------+
|68.64174182577678|
|69.03520359494772|
|68.69092454692314|
|69.18275175838681|
|69.477848085265|
|69.18275175838681|
|69.37948264297226|
|69.477848085265|
|69.13356903724043|
|69.62539624870409|
|70.01885801787503|
|70.41231978704596|
|68.14991461431312|
|70.2155889024605|
|68.98602087380134|
|70.65823339277779|
|68.78928998921589|
|71.44515693111964|
|68.83847271036224|
|68.98602087380134|
+----------------+
only showing top 20 rows
```

（3）关闭 Spark Session

在实际数据科学项目中，通常采用部署/生产模型（Deployment/Productionizing Models）的理论和技术，将 Spark 机器学习模型部署至具体业务系统。

部署/生产模型（Deployment/Productionizing Models）是近几年来广泛受到重视的新兴领域，建议读者学习和关注该领域的最新方法和技术。

在 Spark 数据分析结束时，需要关闭 Spark Session，可以调用 PySpark 提供的 stop() 方法。示例如下。

```
In[33] #关闭 Spark Session
       mySpark.stop()
```