PySpark is a tool created by Apache Spark Community for using Python with Spark. It allows working with RDD (Resilient Distributed Dataset) in Python. It also offers PySpark Shell to link Python APIs with Spark core to initiate Spark Context. Spark is the name engine to realize cluster computing, while PySpark is Python's library to use Spark.

Regular machine learning projects are built around the following methodology:

- Load the data to the disk
- Import the data into the machine's memory
- Process/analyze the data
- Build the machine learning model
- Store the prediction back to disk

Data scientist main's job is to analyze and build predictive models. In short, a data scientist needs to know how to query data using SQL, produce a statistical report and make use of machine learning to produce predictions. Data scientist spends a significant amount of their time on cleaning, transforming and analyzing the data. Once the dataset or data workflow is ready, the data scientist uses various techniques to discover insights and hidden patterns. The data manipulation should be robust and the same easy to use. Spark is the right tool thanks to its speed and rich APIs.

How to Install PySpark

```
conda install pyspark
```

Basic operation with PySpark

```
#initialize the SQLContext
import pyspark
from pyspark import SparkContext
from pyspark import SparkFiles

url = https://raw.githubusercontent.com/guru99-edu/R-
Programming/master/adult_data.csv

sc = SparkContext()

sc.addFile(url)
sqlContext = SQLContext(sc)

#read csv file
df = sqlContext.read.csv(SparkFiles.get("adult_data.csv"), header=True,
inferSchema= True)
```

```
df.printSchema()
root
 |-- age: integer (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: integer (nullable = true)
 |-- education: string (nullable = true)
 |-- education_num: integer (nullable = true)
 |-- marital: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital_gain: integer (nullable = true)
 |-- capital_loss: integer (nullable = true)
 |-- hours_week: integer (nullable = true)
 |-- native_country: string (nullable = true)
 |-- label: string (nullable = true)
```

```
COLUMNS = ['age', 'age_square', 'workclass', 'fnlwgt', 'education',
'education_num', 'marital',
           'occupation', 'relationship', 'race', 'sex', 'capital_gain',
'capital_loss',
           'hours_week', 'native_country', 'label']
df = df.select(COLUMNS)
df.first()

Row(age=39, age_square=1521.0, workclass='State-gov', fnlwgt=77516,
education='Bachelors', education_num=13, marital='Never-married',
occupation='Adm-clerical', relationship='Not-in-family', race='White',
sex='Male', capital_gain=2174, capital_loss=0, hours_week=40,
native_country='United-States', label='<=50K')
```

Build a data processing pipeline

1. Encode the categorical data

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoderEstimator

CATE_FEATURES = ['workclass', 'education', 'marital', 'occupation',
'relationship', 'race', 'sex', 'native_country']
stages = [] # stages in our Pipeline
for categoricalCol in CATE_FEATURES:
    stringIndexer = StringIndexer(inputCol=categoricalCol,
outputCol=categoricalCol + "Index")
    encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()],
                                     outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]
```

2. Index the label feature

```
# Convert label into label indices using the StringIndexer
label_stringIdx =  StringIndexer(inputCol="label", outputCol="newlabel")
stages += [label_stringIdx]
```

3. Add continuous variable

```
from pyspark.sql.types import *

# Write a custom function to convert the data type of DataFrame columns
def convertColumn(df, names, newType):
    for name in names:
        df = df.withColumn(name, df[name].cast(newType))
    return df
# List of continuous features
CONTI_FEATURES  = ['age', 'fnlwgt','capital_gain', 'education_num',
'capital_loss', 'hours_week']
# Convert the type
df_string = convertColumn(df, CONTI_FEATURES, FloatType())

assemblerInputs = [c + "classVec" for c in CATE_FEATURES] + CONTI_FEATURES
```

4. Assemble the steps.

```
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]

# Create a Pipeline.
pipeline = Pipeline(stages=stages)
pipelineModel = pipeline.fit(df_string)
model = pipelineModel.transform(df_string)
```

Create the train data

```
from pyspark.ml.linalg import DenseVector

input_data = model.rdd.map(lambda x: (x["newlabel"],
DenseVector(x["features"])))

df_train = sqlContext.createDataFrame(input_data, ["label", "features"])
```