Learning Apache Spark with Python

Release v1.0

Wenqiang Feng

February 15, 2019

Welcome to my **Learning Apache Spark with Python** note! In this note, you will learn a wide array of concepts about **PySpark** in Data Mining, Text Mining, Machine Leanring and Deep Learning. The PDF version can be downloaded from HERE.

CHAPTER

ONE

PREFACE

1.1 About

1.1.1 About this note

This is a shared repository for Learning Apache Spark Notes. The first version was posted on Github in ChenFeng ([Feng2017]). This shared repository mainly contains the self-learning and self-teaching notes from Wenqiang during his IMA Data Science Fellowship. The reader is referred to the repository https://github.com/runawayhorse001/LearningApacheSpark for more details about the `dataset` and the `.ipynb` files.

In this repository, I try to use the detailed demo code and examples to show how to use each main functions. If you find your work wasn't cited in this note, please feel free to let me know.

Although I am by no means an data mining programming and Big Data expert, I decided that it would be useful for me to share what I learned about PySpark programming in the form of easy tutorials with detailed example. I hope those tutorials will be a valuable tool for your studies.

The tutorials assume that the reader has a preliminary knowledge of programing and Linux. And this document is generated automatically by using sphinx.

1.1.2 About the authors

- **Wenqiang Feng**
  - Data Scientist and PhD in Mathematics
  - University of Tennessee at Knoxville
  - Email: von198@gmail.com

- **Biography**

  Wenqiang Feng is Data Scientist within DST's Applied Analytics Group. Dr. Feng's responsibilities include providing DST clients with access to cutting-edge skills and technologies, including Big Data analytic solutions, advanced analytic and data enhancement techniques and modeling.

  Dr. Feng has deep analytic expertise in data mining, analytic systems, machine learning algorithms, business intelligence, and applying Big Data tools to strategically solve industry problems in a cross-functional business. Before joining DST, Dr. Feng was an IMA Data Science Fellow at The Institute for Mathematics and its Applications (IMA) at the University of Minnesota. While there, he helped startup companies make marketing decisions based on deep predictive analytics.

Dr. Feng graduated from University of Tennessee, Knoxville, with Ph.D. in Computational Mathematics and Master's degree in Statistics. He also holds Master's degree in Computational Mathematics from Missouri University of Science and Technology (MST) and Master's degree in Applied Mathematics from the University of Science and Technology of China (USTC).

- **Declaration**

  The work of Wenqiang Feng was supported by the IMA, while working at IMA. However, any opinion, finding, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the IMA, UTK and DST.

1.2 Motivation for this tutorial

I was motivated by the IMA Data Science Fellowship project to learn PySpark. After that I was impressed and attracted by the PySpark. And I foud that:

1. It is no exaggeration to say that Spark is the most powerful Bigdata tool.
2. However, I still found that learning Spark was a difficult process. I have to Google it and identify which one is true. And it was hard to find detailed examples which I can easily learned the full process in one file.
3. Good sources are expensive for a graduate student.

1.3 Copyright notice and license info

This Learning Apache Spark with Python PDF file is supposed to be a free and living document, which is why its source is available online at https://runawayhorse001.github.io/LearningApacheSpark/pyspark.pdf. But this document is licensed according to both MIT License and Creative Commons Attribution-NonCommercial 2.0 Generic (CC BY-NC 2.0) License.

When you plan to use, copy, modify, merge, publish, distribute or sublicense, Please see the terms of those licenses for more details and give the corresponding credits to the author.

1.4 Acknowledgement

At here, I would like to thank Ming Chen, Jian Sun and Zhongbo Li at the University of Tennessee at Knoxville for the valuable disscussion and thank the generous anonymous authors for providing the detailed solutions and source code on the internet. Without those help, this repository would not have been possible to be made. Wenqiang also would like to thank the Institute for Mathematics and Its Applications (IMA) at University of Minnesota, Twin Cities for support during his IMA Data Scientist Fellow visit.

A special thank you goes to Dr. Haiping Lu, Lecturer in Machine Learning at Department of Computer Science, University of Sheffield, for recommending and heavily using my tutorial in his teaching class and for the valuable suggestions.

1.5 Feedback and suggestions

Your comments and suggestions are highly appreciated. I am more than happy to receive corrections, suggestions or feedbacks through email (von198@gmail.com) for improvements.

CHAPTER

TWO

WHY SPARK WITH PYTHON ?

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb

I want to answer this question from the following two parts:

2.1 Why Spark?

I think the following four main reasons from Apache Spark™ official website are good enough to convince you to use Spark.

1. Speed

   Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

   Apache Spark has an advanced DAG execution engine that supports acyclic data flow and in-memory computing.

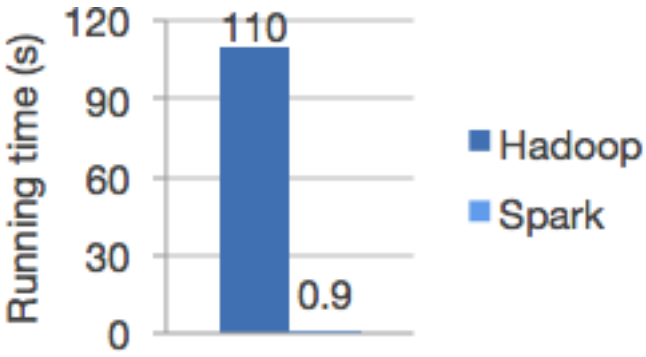


Figure 2.1: Logistic regression in Hadoop and Spark

2. Ease of Use

   Write applications quickly in Java, Scala, Python, R.

   Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python and R shells.

3. Generality

   Combine SQL, streaming, and complex analytics.

Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.

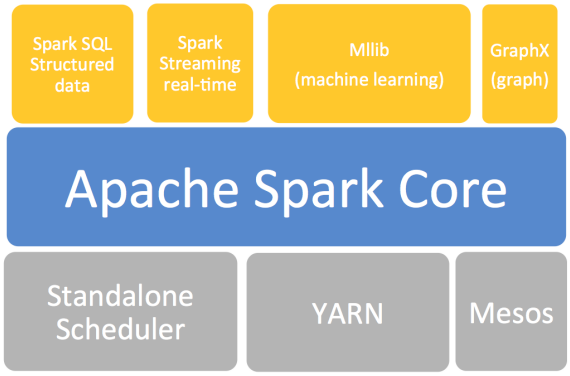


Figure 2.2: The Spark stack

4. Runs Everywhere

   Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

Figure 2.3: The Spark platform

2.2 Why Spark with Python (PySpark)?

No matter you like it or not, Python has been one of the most popular programming languages.

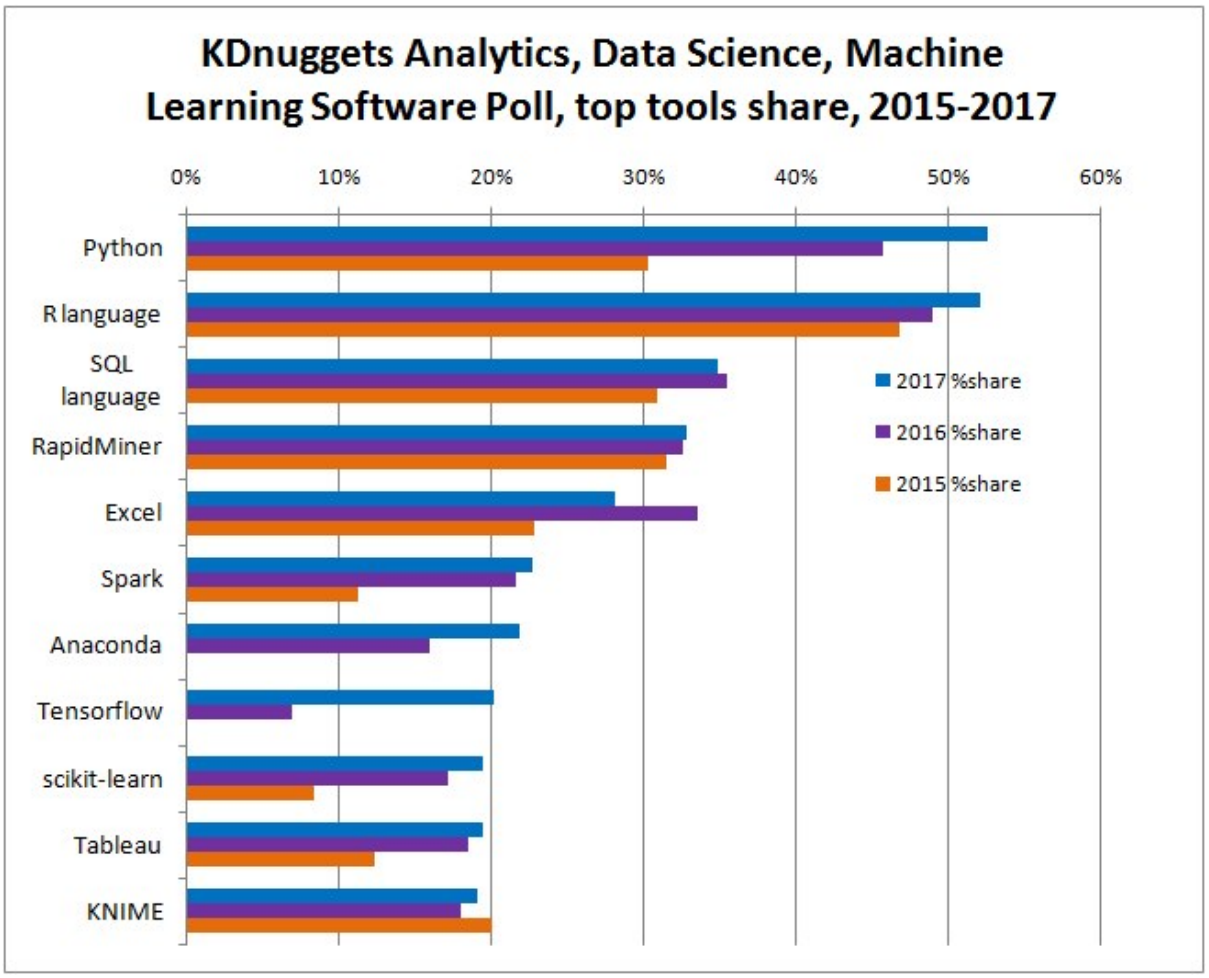


Figure 2.4: KDnuggets Analytics/Data Science 2017 Software Poll from kdnuggets.

CHAPTER

THREE

CONFIGURE RUNNING PLATFORM

Note: Good tools are prerequisite to the successful execution of a job. – old Chinese proverb

A good programming platform can save you lots of troubles and time. Herein I will only present how to install my favorite programming platform and only show the easiest way which I know to set it up on Linux system. If you want to install on the other operator system, you can Google it. In this section, you may learn how to set up Pyspark on the corresponding programming platform and package.

3.1 Run on Databricks Community Cloud

If you don't have any experience with Linux or Unix operator system, I would love to recommend you to use Spark on Databricks Community Cloud. Since you do not need to setup the Spark and it's totally **free** for Community Edition. Please follow the steps listed below.

1. Sign up a account at: https://community.cloud.databricks.com/login.html

2. Sign in with your account, then you can creat your cluster(machine), table(dataset) and notebook(code).

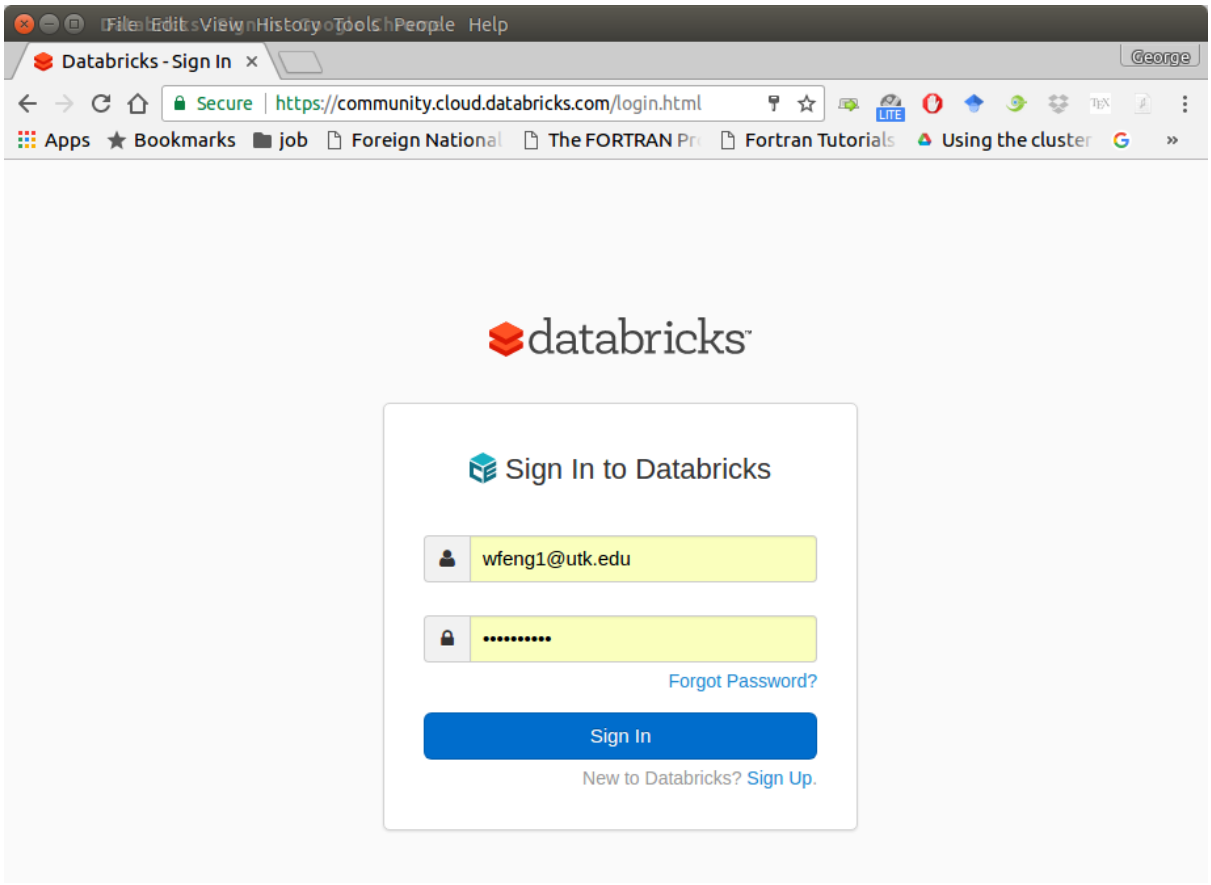
3. Create your cluster where your code will run

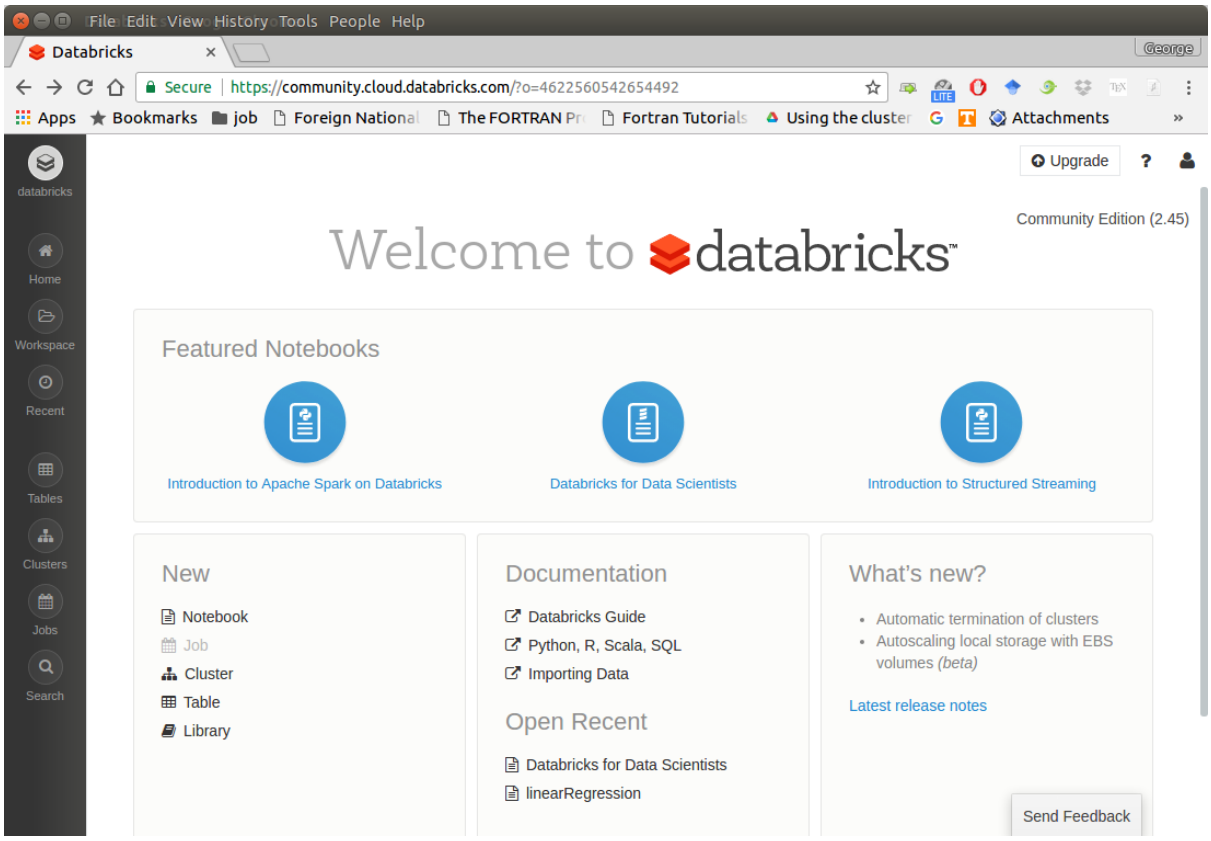
4. Import your dataset

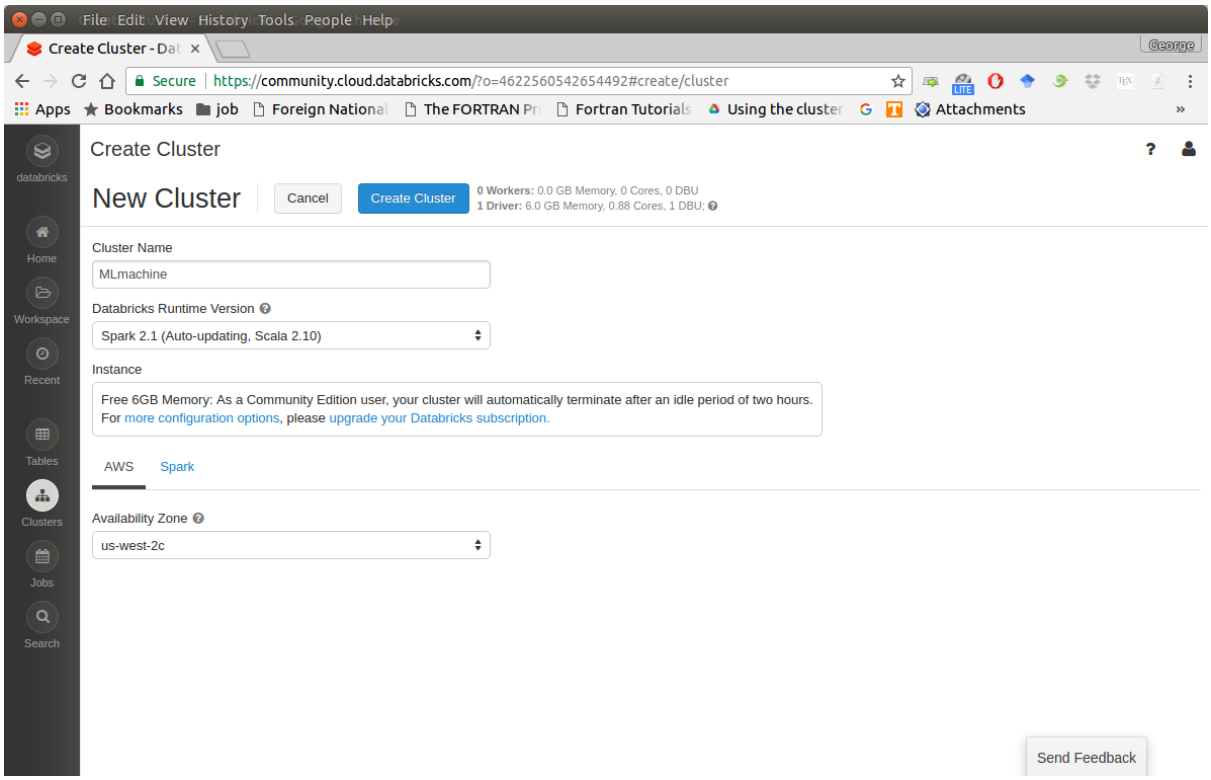
   Note: You need to save the path which appears at Uploaded to DBFS: /FileStore/tables/05rmhuqv1489687378010/. Since we will use this path to load the dataset.

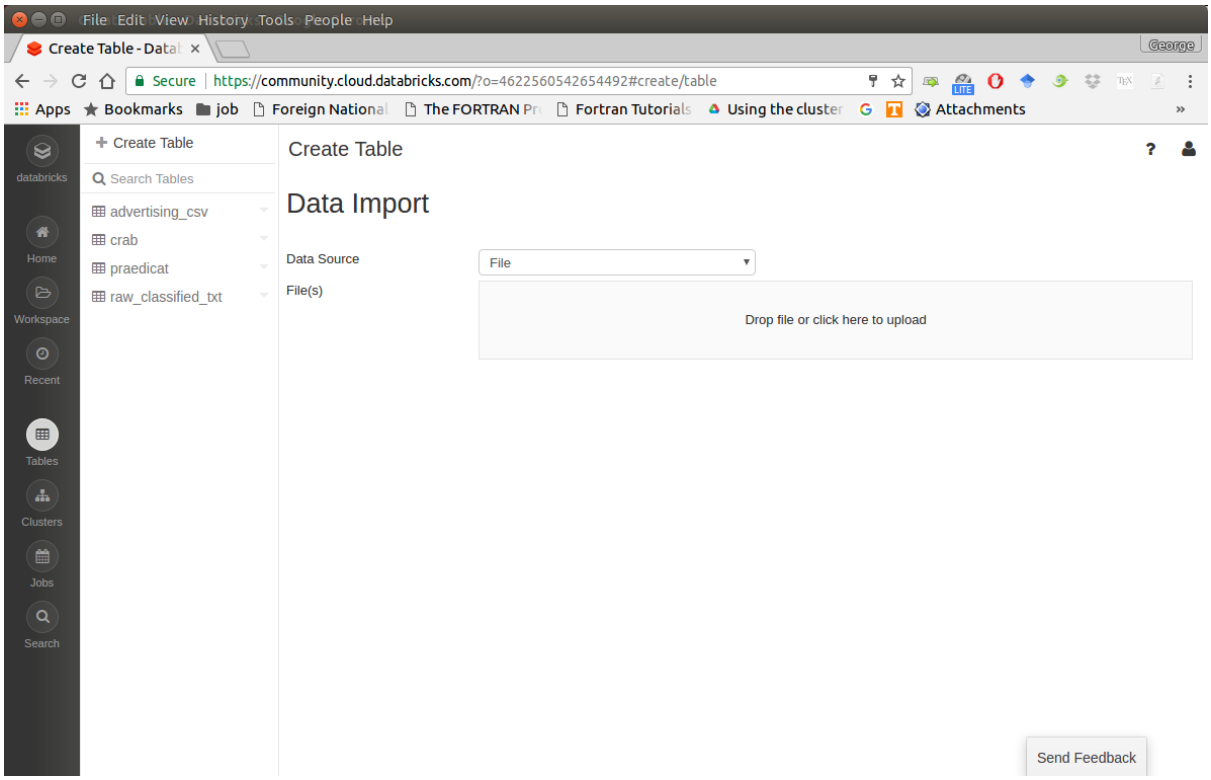
5. Creat your notebook

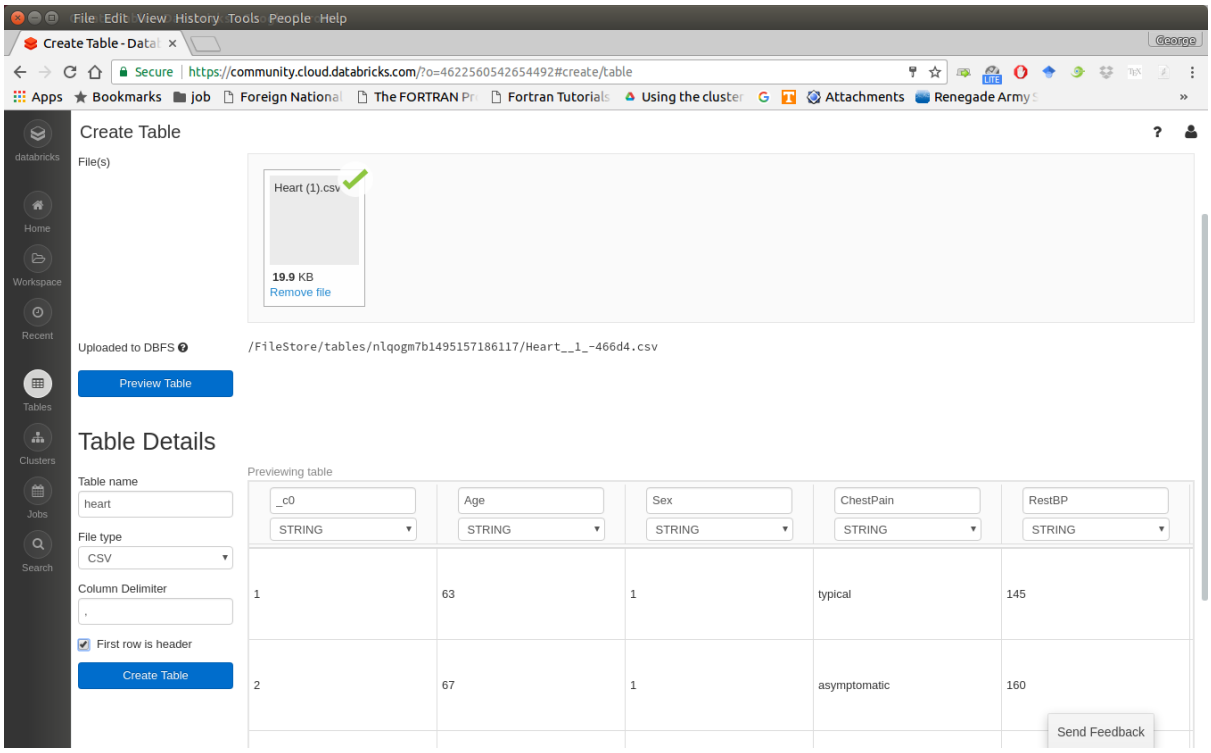
After finishing the above 5 steps, you are ready to run your Spark code on Databricks Community Cloud. I will run all the following demos on Databricks Community Cloud. Hopefully, when you run the demo code, you will get the following results:

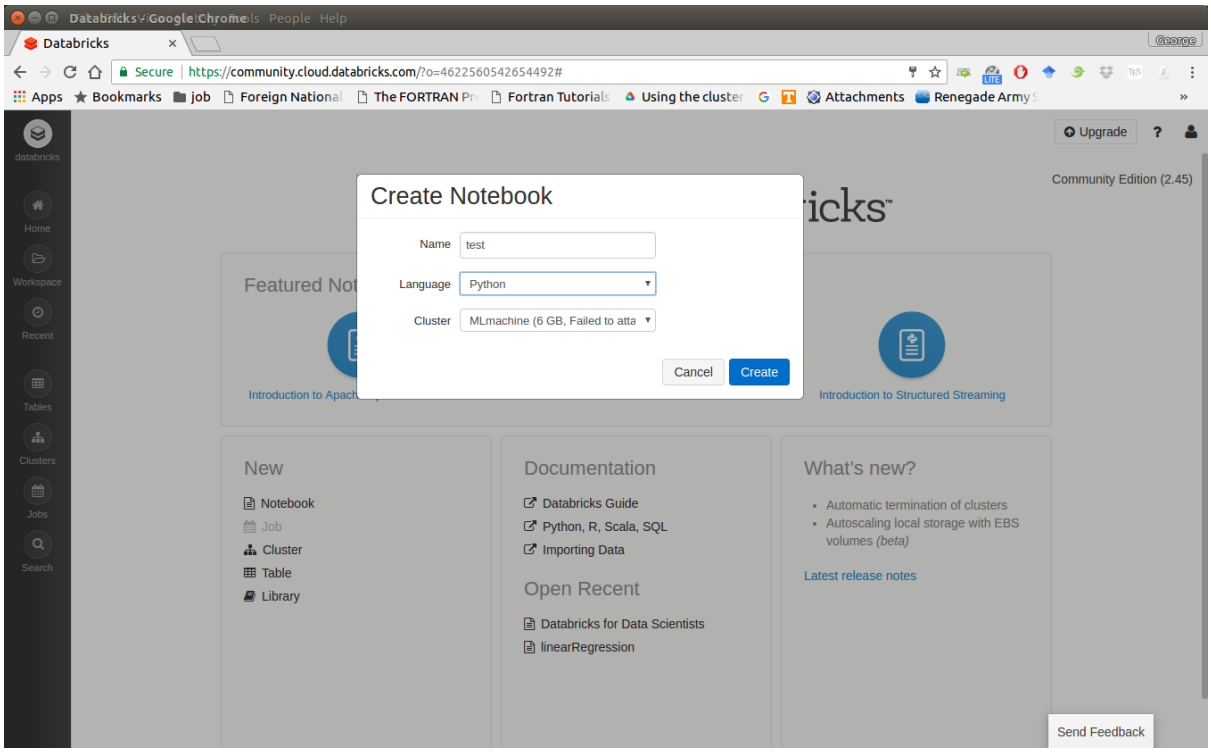

File Edit View History Tools People Help
Databricks - Sign In
George
Secure | https://community.cloud.databricks.com/login.html
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster
databricks
Sign In to Databricks
wfeng1@utk.edu
Forgot Password?
Sign In
New to Databricks? Sign Up.

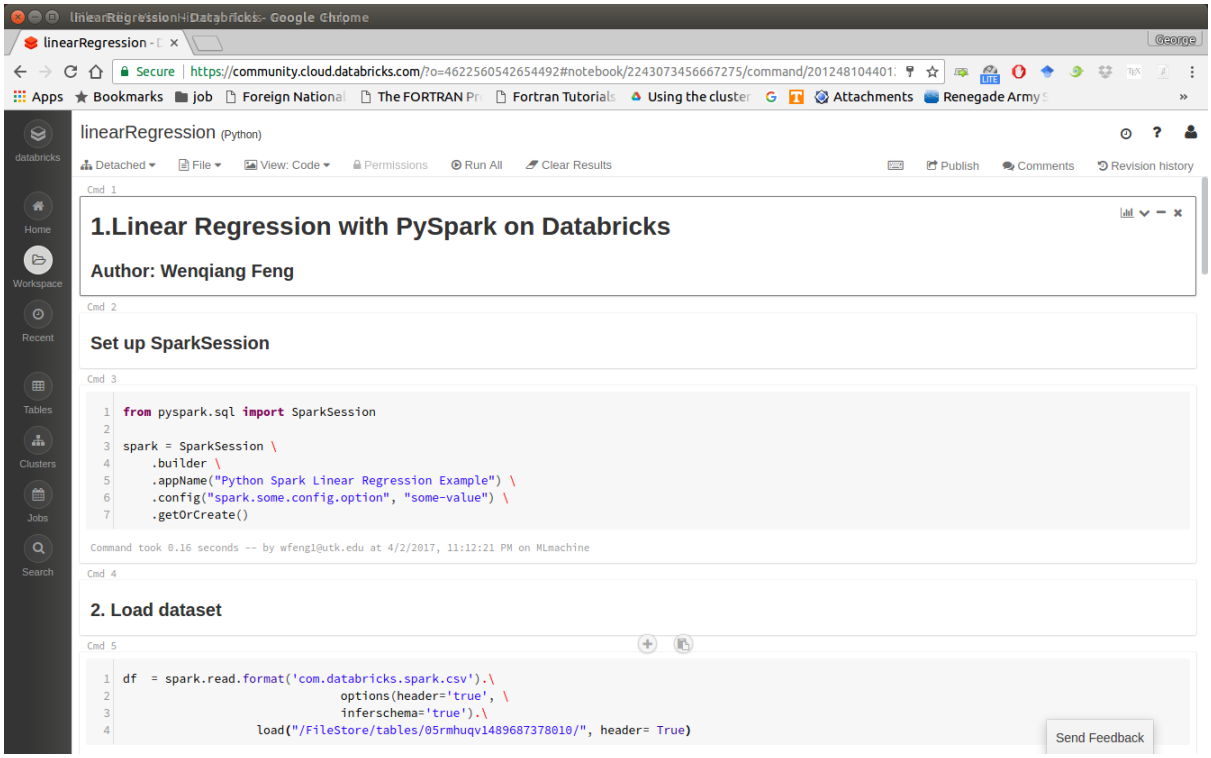

File Edit View History Tools People Help
Databricks
George
Secure | https://community.cloud.databricks.com/?o=4622560542654492
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments
databricks
Home
Workspace
Recent
Tables
Clusters
Jobs
Search
Upgrade
Community Edition (2.45)
Welcome to databricks
Featured Notebooks
Introduction to Apache Spark on Databricks
Databricks for Data Scientists
Introduction to Structured Streaming
New
Notebook
Job
Cluster
Table
Library
Documentation
Databricks Guide
Python, R, Scala, SQL
Importing Data
Open Recent
Databricks for Data Scientists
linearRegression
What's new?
Automatic termination of clusters
Autoscaling local storage with EBS volumes (beta)
Latest release notes
Send Feedback


File Edit View History Tools People Help
Create Cluster - Dat
George
Secure | https://community.cloud.databricks.com/?o=4622560542654492#create/cluster
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments
databricks
Home
Workspace
Recent
Tables
Clusters
Jobs
Search
Create Cluster
New Cluster
Cancel
Create Cluster
0 Workers: 0.0 GB Memory, 0 Cores, 0 DBU
1 Driver: 6.0 GB Memory, 0.88 Cores, 1 DBU;
Cluster Name
MLmachine
Databricks Runtime Version
Spark 2.1 (Auto-updating, Scala 2.10)
Instance
Free 6GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours.
For more configuration options, please upgrade your Databricks subscription.
AWS
Spark
Availability Zone
us-west-2c
Send Feedback


File Edit View History Tools People Help
Create Table - Datal
George
Secure | https://community.cloud.databricks.com/?o=4622560542654492#create/table
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments
databricks
Home
Workspace
Recent
Tables
Clusters
Jobs
Search
Create Table
Search Tables
advertising_csv
crab
praedicat
raw_classified_txt
Create Table
Data Import
Data Source
File
File(s)
Drop file or click here to upload
Send Feedback


Create Table
File(s)
Heart (1).csv
19.9 KB
Remove file
Uploaded to DBFS
/FileStore/tables/nlqogm7b1495157186117/Heart__1_-466d4.csv
Preview Table
Table Details
Table name
heart
File type
CSV
Column Delimiter
,
First row is header
Create Table
Previewing table
_c0
Age
Sex
ChestPain
RestBP
STRING
1
63
1
typical
145
2
67
1
asymptomatic
160
Send Feedback


Create Notebook
Name
test
Language
Python
Cluster
MLmachine (6 GB, Failed to atta
Cancel
Create
Upgrade
Community Edition (2.45)
Introduction to Structured Streaming
New
Notebook
Job
Cluster
Table
Library
Documentation
Databricks Guide
Python, R, Scala, SQL
Importing Data
Open Recent
Databricks for Data Scientists
linearRegression
What's new?
Automatic termination of clusters
Autoscaling local storage with EBS volumes (beta)
Latest release notes
Send Feedback




```
+---+-----+-----+---------+-----+
|_c0|   TV|Radio|Newspaper|Sales|
+---+-----+-----+---------+-----+
|  1|230.1| 37.8|     69.2| 22.1|
|  2| 44.5| 39.3|     45.1| 10.4|
|  3| 17.2| 45.9|     69.3|  9.3|
|  4|151.5| 41.3|     58.5| 18.5|
|  5|180.8| 10.8|     58.4| 12.9|
+---+-----+-----+---------+-----+
only showing top 5 rows

root
 |-- _c0: integer (nullable = true)
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

3.2 Configure Spark on Mac and Ubuntu

3.2.1 Installing Prerequisites

I will strongly recommend you to install Anaconda, since it contains most of the prerequisites and support multiple Operator Systems.

1. **Install Python**

Go to Ubuntu Software Center and follow the following steps:

1. Open Ubuntu Software Center

2. Search for python
3. And click Install

Or Open your terminal and using the following command:

```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev
                libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev
sudo apt-get install python
sudo easy_install pip
sudo pip install ipython
```

3.2.2 Install Java

Java is used by many other softwares. So it is quite possible that you have already installed it. You can by using the following command in Command Prompt:

```
java -version
```

Otherwise, you can follow the steps in How do I install Java for my Mac? to install java on Mac and use the following command in Command Prompt to install on Ubuntu:

```
sudo apt-add-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

3.2.3 Install Java SE Runtime Environment

I installed ORACLE Java JDK.

Warning: Installing Java and Java SE Runtime Environment steps are very important, since Spark is a domain-specific language written in Java.

You can check if your Java is available and find it's version by using the following command in Command Prompt:

```
java -version
```

If your Java is installed successfully, you will get the similar results as follows:

```
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

3.2.4 Install Apache Spark

Actually, the Pre-build version doesn't need installation. You can use it when you unpack it.

1. Download: You can get the Pre-built Apache Spark™ from Download Apache Spark™.
2. Unpack: Unpack the Apache Spark™ to the path where you want to install the Spark.

3. Test: Test the Prerequisites: change the direction `spark-#.#.#-bin-hadoop#.#/bin` and run

```
./pyspark
```

```
Python 2.7.13 |Anaconda 4.4.0 (x86_64)| (default, Dec 20 2016, 23:05:08)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR,
use setLogLevel(newLevel).
17/08/30 13:30:12 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
17/08/30 13:30:17 WARN ObjectStore: Failed to get database global_temp,
returning NoSuchObjectException
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.1
      /_/

Using Python version 2.7.13 (default, Dec 20 2016 23:05:08)
SparkSession available as 'spark'.
```

3.2.5 Configure the Spark

1. **Mac Operator System:** open your `bash_profile` in Terminal

```
vim ~/.bash_profile
```

And add the following lines to your `bash_profile` (remember to change the path)

```
# add for spark
export SPARK_HOME=your_spark_installation_path
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PATH=$PATH:$SPARK_HOME/bin
export PYSPARK_DRIVE_PYTHON="jupyter"
export PYSPARK_DRIVE_PYTHON_OPTS="notebook"
```

At last, remember to source your `bash_profile`

```
source ~/.bash_profile
```

2. **Ubuntu Operator Sysytem:** open your `bashrc` in Terminal

```
vim ~/.bashrc
```

And add the following lines to your `bashrc` (remember to change the path)

```
# add for spark
export SPARK_HOME=your_spark_installation_path
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PATH=$PATH:$SPARK_HOME/bin
```

```
export PYSPARK_DRIVE_PYTHON="jupyter"
export PYSPARK_DRIVE_PYTHON_OPTS="notebook"
```

At last, remember to source your `bashrc`

```
source ~/.bashrc
```

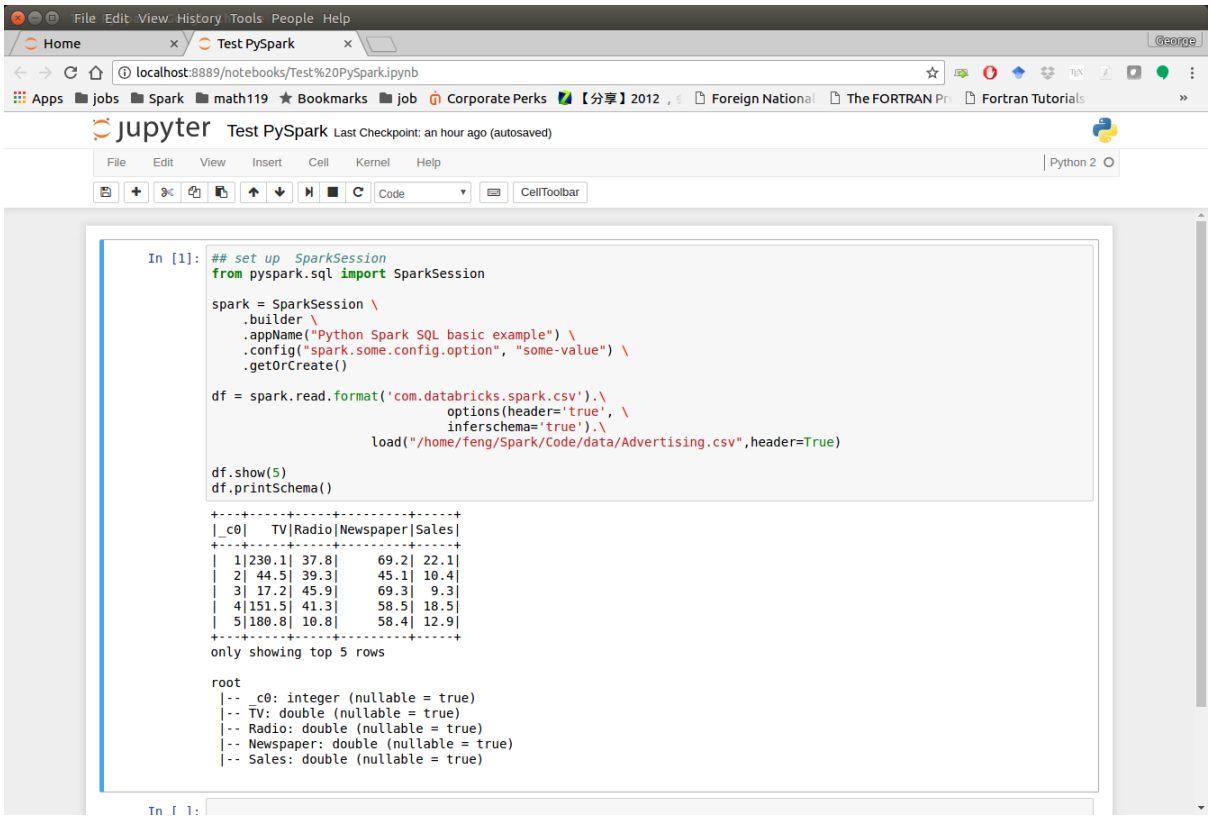
3.3 Configure Spark on Windows

Installing open source software on Windows is always a nightmare for me. Thanks for Deelesh Mandloi. You can follow the detailed procedures in the blog Getting Started with PySpark on Windows to install the Apache Spark™ on your Windows Operator System.

3.4 PySpark With Text Editor or IDE

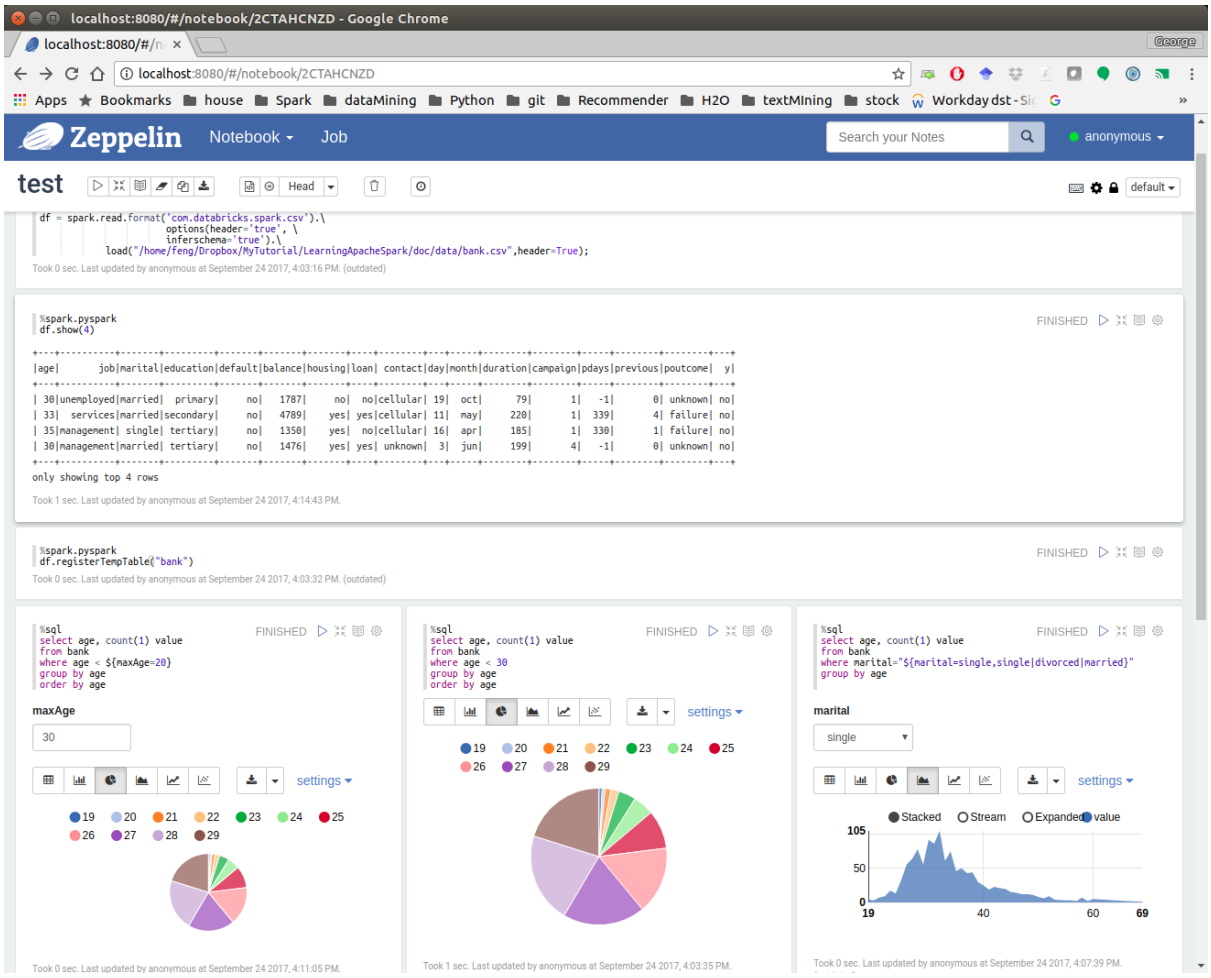
3.4.1 PySpark With Jupyter Notebook

After you finishing the above setup steps in *Configure Spark on Mac and Ubuntu*, then you should be good to write and run your PySpark Code in Jupyter notebook.



3.4.2 PySpark With Apache Zeppelin

After you finishing the above setup steps in *Configure Spark on Mac and Ubuntu*, then you should be good to write and run your PySpark Code in Apache Zeppelin.



3.4.3 PySpark With Sublime Text

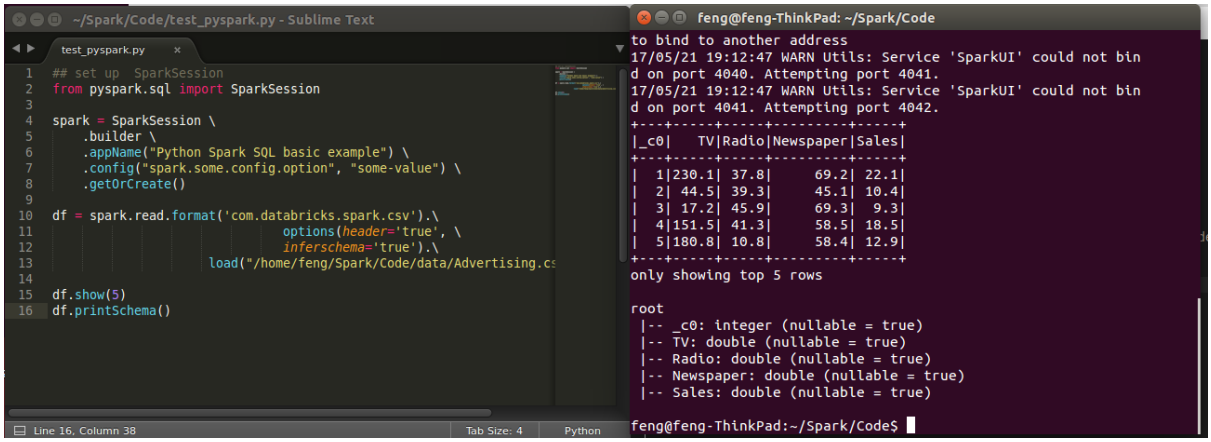
After you finishing the above setup steps in *Configure Spark on Mac and Ubuntu*, then you should be good to use Sublime Text to write your PySpark Code and run your code as a normal python code in Terminal.

```
python test_pyspark.py
```

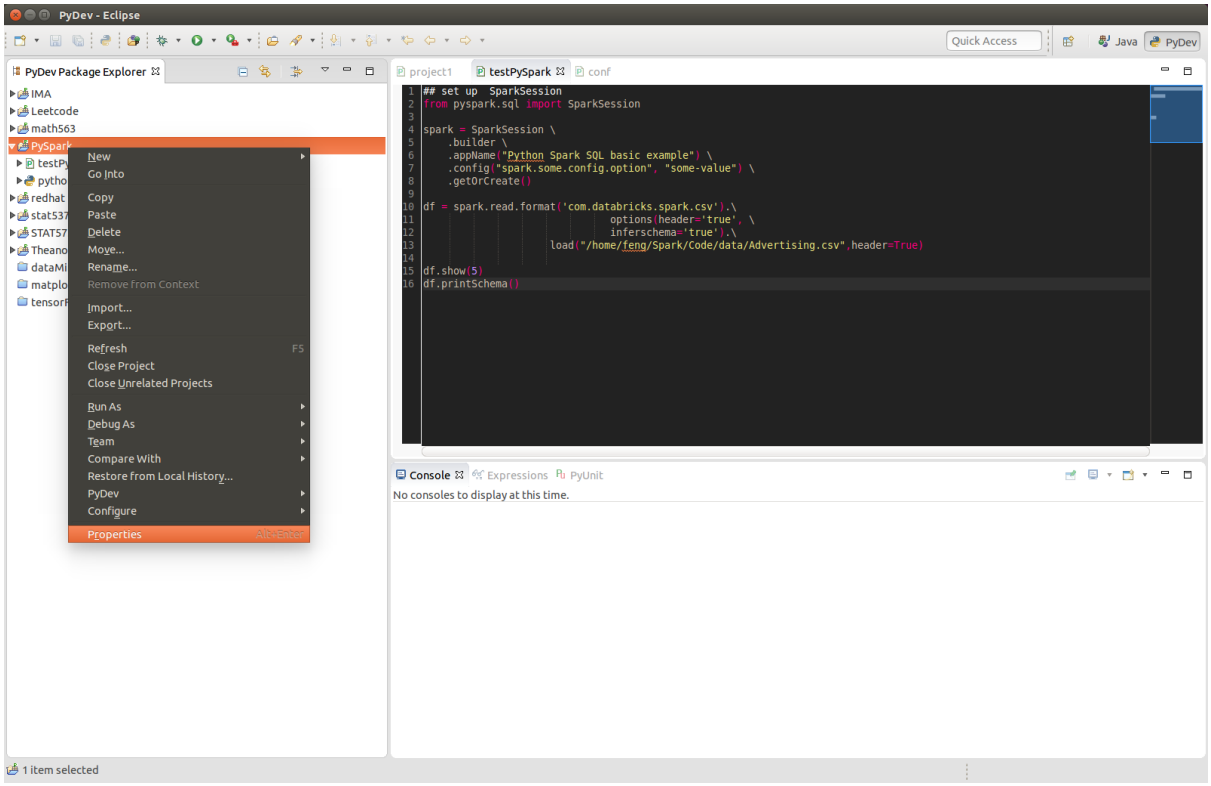
Then you should get the output results in your terminal.

3.4.4 PySpark With Eclipse

If you want to run PySpark code on Eclipse, you need to add the paths for the **External Libraries** for your **Current Project** as follows:



1. Open the properties of your project

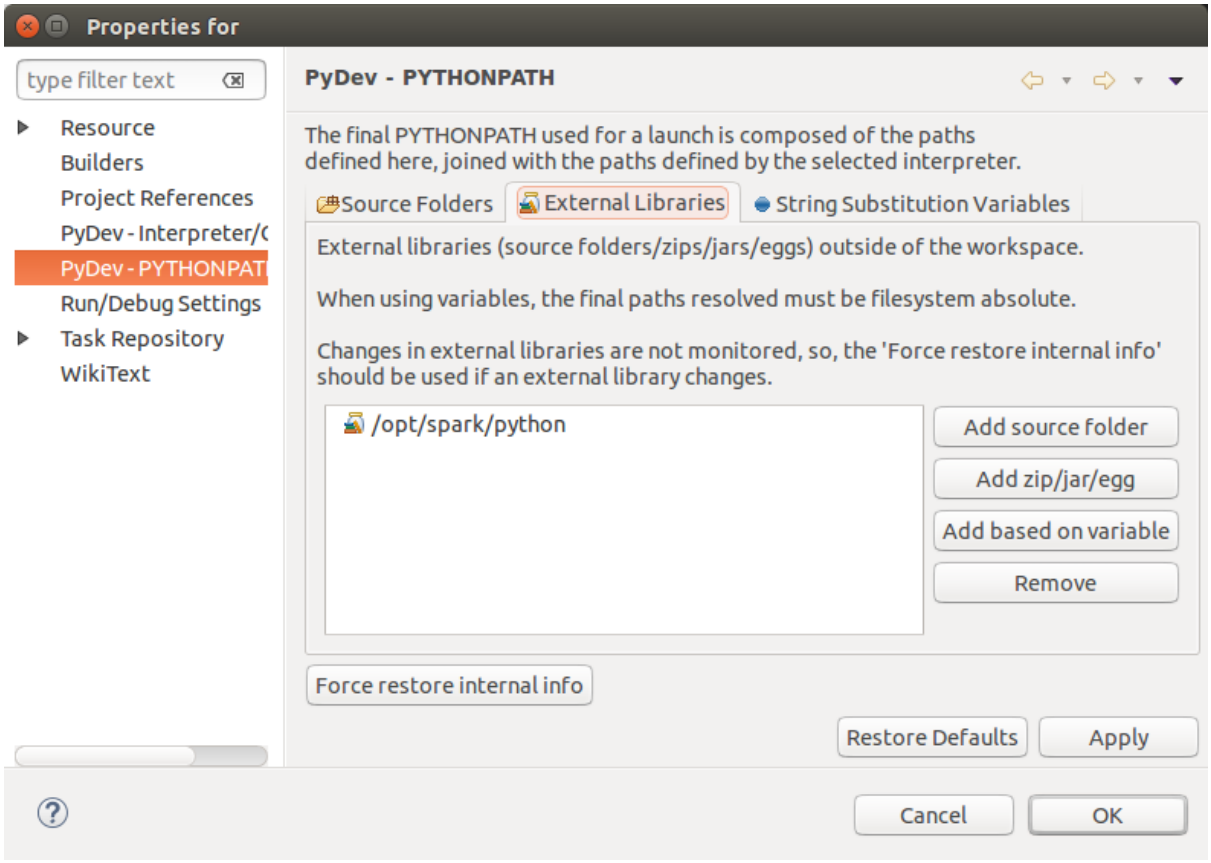


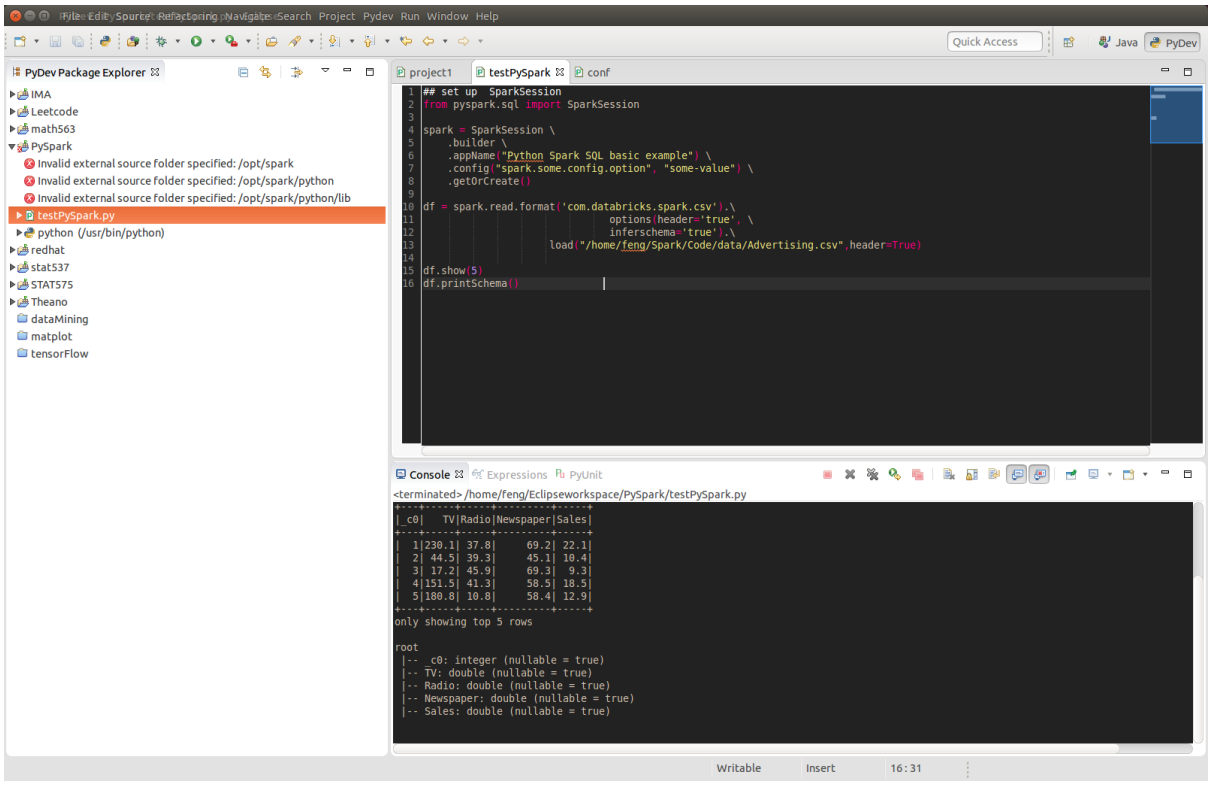
2. Add the paths for the **External Libraries**

And then you should be good to run your code on Eclipse with PyDev.

3.5 Set up Spark on Cloud

Following the setup steps in *Configure Spark on Mac and Ubuntu*, you can set up your own cluster on the cloud, for example AWS, Google Cloud. Actually, for those clouds, they have their own Big Data


Properties for
type filter text
Resource
Builders
Project References
PyDev - Interpreter/(
PyDev - PYTHONPAT
Run/Debug Settings
Task Repository
WikiText
PyDev - PYTHONPATH
The final PYTHONPATH used for a launch is composed of the paths
defined here, joined with the paths defined by the selected interpreter.
Source Folders
External Libraries
String Substitution Variables
External libraries (source folders/zips/jars/eggs) outside of the workspace.
When using variables, the final paths resolved must be filesystem absolute.
Changes in external libraries are not monitored, so, the 'Force restore internal info'
should be used if an external library changes.
/opt/spark/python
Add source folder
Add zip/jar/egg
Add based on variable
Remove
Force restore internal info
Restore Defaults
Apply
Cancel
OK


PyDev Package Explorer
IMA
Leetcode
math563
PySpark
Invalid external source folder specified: /opt/spark
Invalid external source folder specified: /opt/spark/python
Invalid external source folder specified: /opt/spark/python/lib
testPySpark.py
python (/usr/bin/python)
redhat
stat537
STAT575
Theano
dataMining
matplot
tensorFlow
project1
testPySpark
conf
set up SparkSession
from pyspark.sql import SparkSession
spark = SparkSession \
.builder \
.appName("Python Spark SQL basic example") \
.config("spark.some.config.option", "some-value") \
.getOrCreate()
df = spark.read.format('com.databricks.spark.csv').\
options(header='true', \
inferschema='true').\
load("/home/feng/Spark/Code/data/Advertising.csv",header=True)
df.show(5)
df.printSchema()
Console
<terminated> /home/feng/Eclipseworkspace/PySpark/testPySpark.py
|_c0| TV|Radio|Newspaper|Sales|
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9
only showing top 5 rows
root
-- _c0: integer (nullable = true)
-- TV: double (nullable = true)
-- Radio: double (nullable = true)
-- Newspaper: double (nullable = true)
-- Sales: double (nullable = true)
Writable
Insert
16:31

tool. Yon can run them directly whitout any setting just like Databricks Community Cloud. If you want more details, please feel free to contact with me.

3.6 Demo Code in this Section

The code for this section is available for download test_pyspark, and the Jupyter notebook can be download from test_pyspark_ipynb.

- Python Source code

```
## set up  SparkSession
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

df = spark.read.format('com.databricks.spark.csv').\
                                options(header='true', \
                                inferschema='true').\
                    load("/home/feng/Spark/Code/data/Advertising.csv",header=True)

df.show(5)
df.printSchema()
```

CHAPTER

FOUR

AN INTRODUCTION TO APACHE SPARK

Note: **Know yourself and know your enemy, and you will never be defeated** – idiom, from Sunzi's Art of War

4.1 Core Concepts

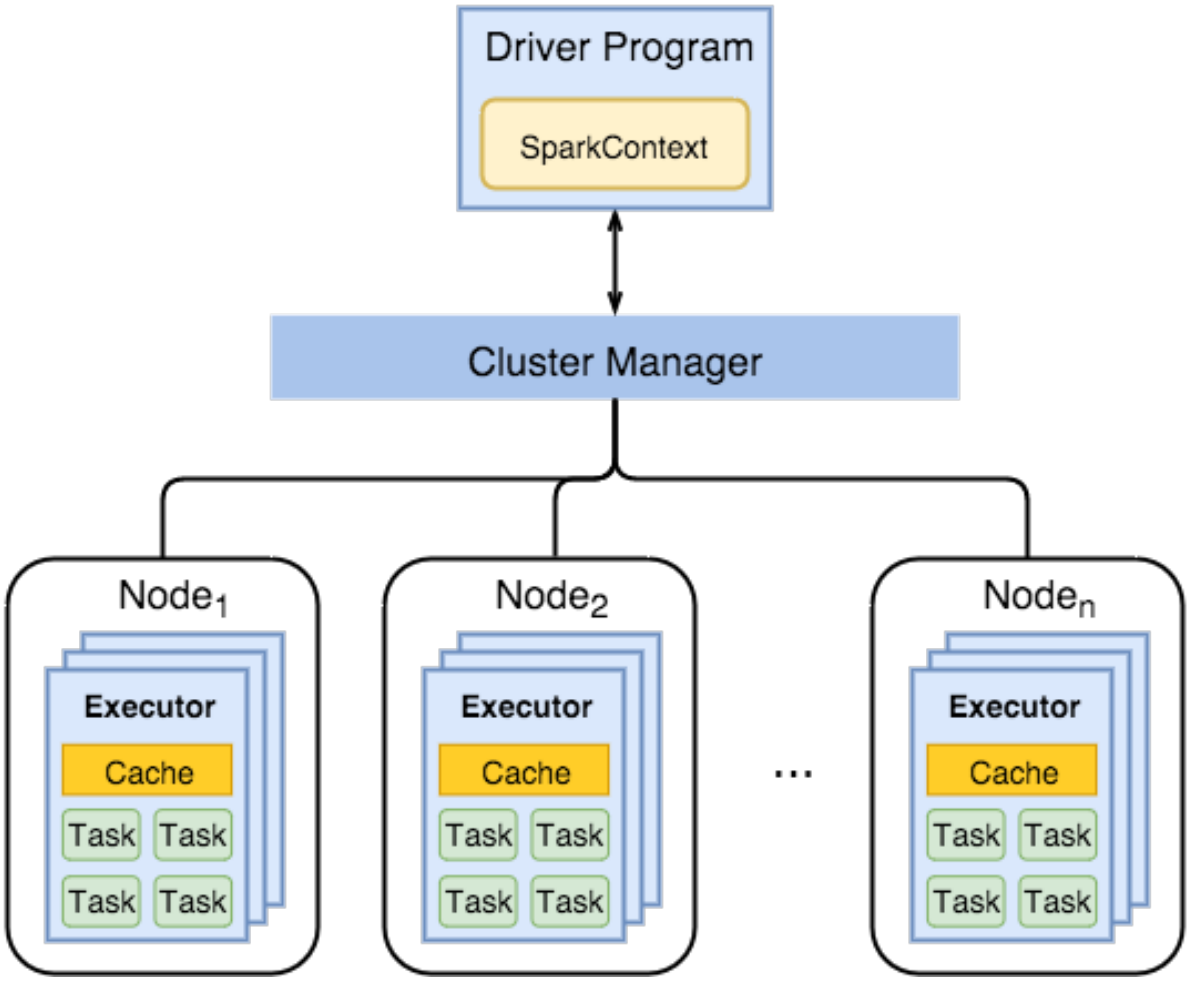
Most of the following content comes from [Kirillov2016]. So the copyright belongs to **Anton Kirillov**. I will refer you to get more details from Apache Spark core concepts, architecture and internals.

Before diving deep into how Apache Spark works, lets understand the jargon of Apache Spark

- Job: A piece of code which reads some input from HDFS or local, performs some computation on the data and writes some output data.
- Stages: Jobs are divided into stages. Stages are classified as a Map or reduce stages (Its easier to understand if you have worked on Hadoop and want to correlate). Stages are divided based on computational boundaries, all computations (operators) cannot be Updated in a single Stage. It happens over many stages.
- Tasks: Each stage has some tasks, one task per partition. One task is executed on one partition of data on one executor (machine).
- DAG: DAG stands for Directed Acyclic Graph, in the present context its a DAG of operators.
- Executor: The process responsible for executing a task.
- Master: The machine on which the Driver program runs
- Slave: The machine on which the Executor program runs

4.2 Spark Components

1. Spark Driver
   - separate process to execute user applications
   - creates SparkContext to schedule jobs execution and negotiate with cluster manager
2. Executors
   - run tasks scheduled by driver

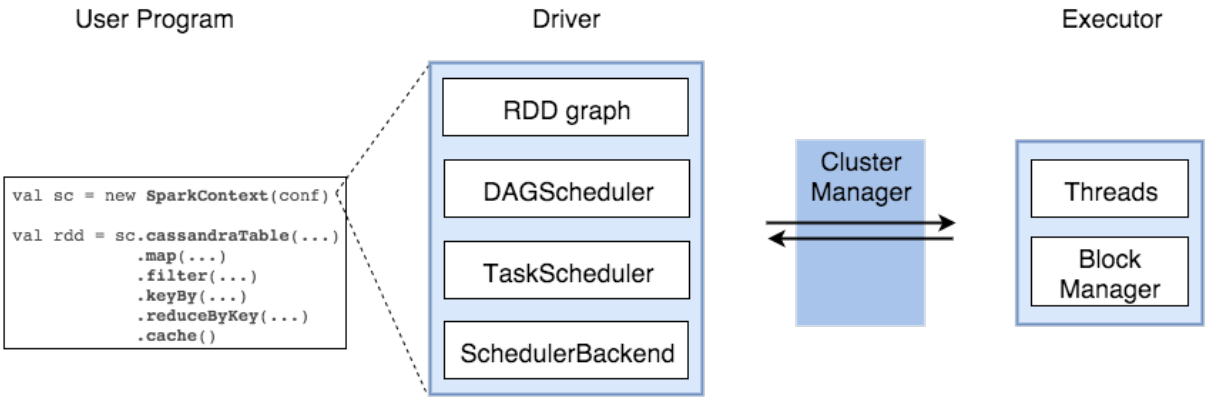

Driver Program
SparkContext
Cluster Manager
Node1
Node2
Noden
Executor
Cache
Task
Task
Task
Task
Executor
Cache
Task
Task
Task
Task
...
Executor
Cache
Task
Task
Task
Task

- store computation results in memory, on disk or off-heap
- interact with storage systems

3. Cluster Manager

- Mesos
- YARN
- Spark Standalone

Spark Driver contains more components responsible for translation of user code into actual jobs executed on cluster:



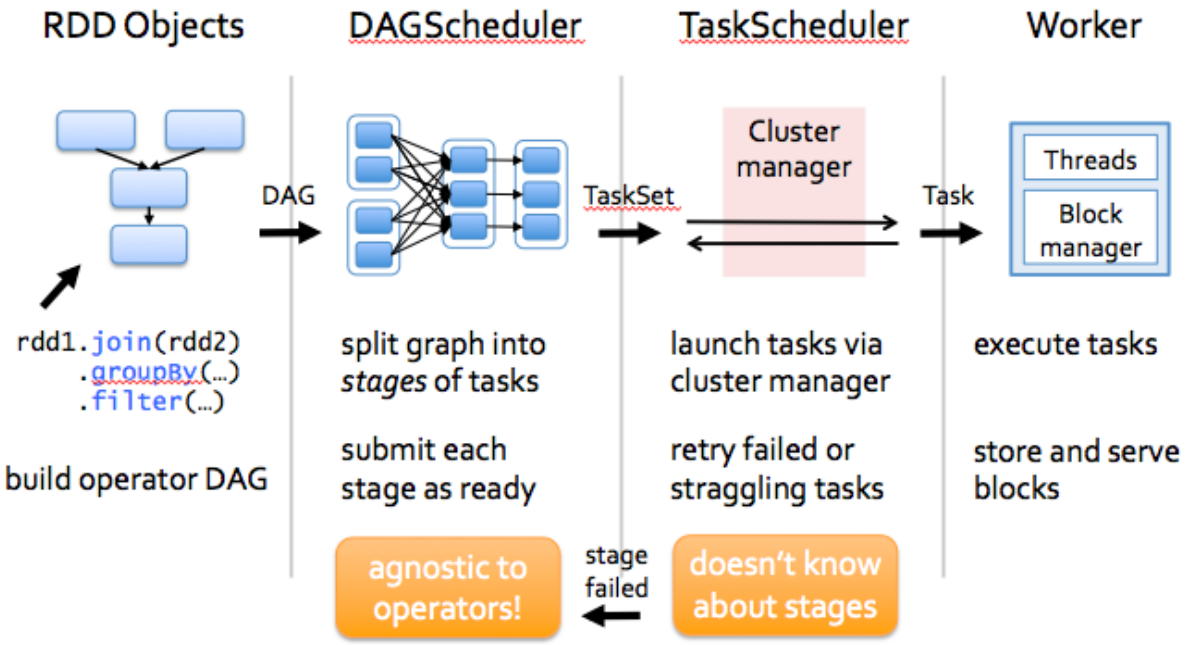
- SparkContext
  - represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster
- DAGScheduler
  - computes a DAG of stages for each job and submits them to TaskScheduler determines preferred locations for tasks (based on cache status or shuffle files locations) and finds minimum schedule to run the jobs
- TaskScheduler
  - responsible for sending tasks to the cluster, running them, retrying if there are failures, and mitigating stragglers
- SchedulerBackend
  - backend interface for scheduling systems that allows plugging in different implementations(Mesos, YARN, Standalone, local)
- BlockManager
  - provides interfaces for putting and retrieving blocks both locally and remotely into various stores (memory, disk, and off-heap)

4.3 Architecture

4.4 How Spark Works?

Spark has a small code base and the system is divided in various layers. Each layer has some responsibilities. The layers are independent of each other.

The first layer is the interpreter, Spark uses a Scala interpreter, with some modifications. As you enter your code in spark console (creating RDD's and applying operators), Spark creates a operator graph. When the user runs an action (like collect), the Graph is submitted to a DAG Scheduler. The DAG scheduler divides operator graph into (map and reduce) stages. A stage is comprised of tasks based on partitions of the input data. The DAG scheduler pipelines operators together to optimize the graph. For e.g. Many map operators can be scheduled in a single stage. This optimization is key to Sparks performance. The final result of a DAG scheduler is a set of stages. The stages are passed on to the Task Scheduler. The task scheduler launches tasks via cluster manager. (Spark Standalone/Yarn/Mesos). The task scheduler doesn't know about dependencies among stages.



CHAPTER

FIVE

PROGRAMMING WITH RDDS

Note: If you only know yourself, but not your opponent, you may win or may lose. If you know neither yourself nor your enemy, you will always endanger yourself – idiom, from Sunzi's Art of War

RDD represents **Resilient Distributed Dataset**. An RDD in Spark is simply an immutable distributed collection of objects sets. Each RDD is split into multiple partitions (similar pattern with smaller sets), which may be computed on different nodes of the cluster.

5.1 Create RDD

Usually, there are two popular way to create the RDDs: loading an external dataset, or distributing a set of collection of objects. The following examples show some simplest ways to create RDDs by using `parallelize()` fucntion which takes an already existing collection in your program and pass the same to the Spark Context.

1. By using `parallelize( )` fucntion

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

df = spark.sparkContext.parallelize([(1, 2, 3, 'a b c'),
              (4, 5, 6, 'd e f'),
              (7, 8, 9, 'g h i')]).toDF(['col1', 'col2', 'col3','col4'])
```

Then you will get the RDD data:

```
df.show()
```

```
+----+----+----+-----+
|col1|col2|col3| col4|
+----+----+----+-----+
|   1|   2|   3|a b c|
|   4|   5|   6|d e f|
|   7|   8|   9|g h i|
+----+----+----+-----+
```

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

myData = spark.sparkContext.parallelize([(1,2), (3,4), (5,6), (7,8), (9,10)])
```

Then you will get the RDD data:

```
myData.collect()

[(1, 2), (3, 4), (5, 6), (7, 8), (9, 10)]
```

2. By using `createDataFrame( )` function

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

Employee = spark.createDataFrame([
                        ('1', 'Joe',   '70000', '1'),
                        ('2', 'Henry', '80000', '2'),
                        ('3', 'Sam',   '60000', '2'),
                        ('4', 'Max',   '90000', '1')],
                        ['Id', 'Name', 'Sallary','DepartmentId']
                       )
```

Then you will get the RDD data:

```
+---+-----+-------+------------+
| Id| Name|Sallary|DepartmentId|
+---+-----+-------+------------+
|  1|  Joe|  70000|           1|
|  2|Henry|  80000|           2|
|  3|  Sam|  60000|           2|
|  4|  Max|  90000|           1|
+---+-----+-------+------------+
```

3. By using `read` and `load` functions

    1. **Read dataset from .csv file**

    ```
    ## set up  SparkSession
    from pyspark.sql import SparkSession

    spark = SparkSession \
        .builder \
        .appName("Python Spark create RDD example") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()

    df = spark.read.format('com.databricks.spark.csv').\
    ```

```
                                    options(header='true', \
                                    inferschema='true').\
                    load("/home/feng/Spark/Code/data/Advertising.csv",header=True)

df.show(5)
df.printSchema()
```

Then you will get the RDD data:

```
+---+-----+-----+---------+-----+
|_c0|   TV|Radio|Newspaper|Sales|
+---+-----+-----+---------+-----+
|  1|230.1| 37.8|     69.2| 22.1|
|  2| 44.5| 39.3|     45.1| 10.4|
|  3| 17.2| 45.9|     69.3|  9.3|
|  4|151.5| 41.3|     58.5| 18.5|
|  5|180.8| 10.8|     58.4| 12.9|
+---+-----+-----+---------+-----+
only showing top 5 rows

root
 |-- _c0: integer (nullable = true)
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

Once created, RDDs offer two types of operations: transformations and actions.

2. **Read dataset from DataBase**

```
## set up  SparkSession
from pyspark.sql import SparkSession

spark = SparkSession \
            .builder \
            .appName("Python Spark create RDD example") \
            .config("spark.some.config.option", "some-value") \
            .getOrCreate()

## User information
user = 'your_username'
pw   = 'your_password'

## Database information
table_name = 'table_name'
url = 'jdbc:postgresql://##.###.###.##:5432/dataset?user='+user+'&password='+pw
properties ={'driver': 'org.postgresql.Driver', 'password': pw,'user': user}

df = spark.read.jdbc(url=url, table=table_name, properties=properties)

df.show(5)
df.printSchema()
```

Then you will get the RDD data:

```
+---+-----+-----+---------+-----+
|_c0|   TV|Radio|Newspaper|Sales|
+---+-----+-----+---------+-----+
```

```
|  1|230.1| 37.8|     69.2| 22.1|
|  2| 44.5| 39.3|     45.1| 10.4|
|  3| 17.2| 45.9|     69.3|  9.3|
|  4|151.5| 41.3|     58.5| 18.5|
|  5|180.8| 10.8|     58.4| 12.9|
+---+-----+-----+---------+-----+
only showing top 5 rows

root
 |-- _c0: integer (nullable = true)
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

Note:

Reading tables from Database needs the proper drive for the corresponding Database. For example, the above demo needs `org.postgresql.Driver` and **you need to download it and put it in "jars" folder of your spark installation path.** I download `postgresql-42.1.1.jar` from the official website and put it in `jars` folder.

3. **Read dataset from HDFS**

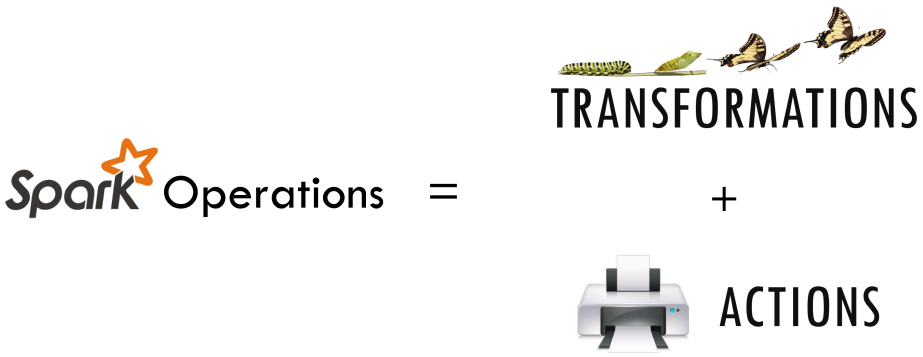
```
from pyspark.conf import SparkConf
from pyspark.context import SparkContext
from pyspark.sql import HiveContext

sc= SparkContext('local','example')
hc = HiveContext(sc)
tf1 = sc.textFile("hdfs://cdhstltest/user/data/demo.CSV")
print(tf1.first())

hc.sql("use intg_cme_w")
spf = hc.sql("SELECT * FROM spf LIMIT 100")
print(spf.show(5))
```

5.2 Spark Operations

Warning: All the figures below are from Jeffrey Thompson. The interested reader is referred to pyspark pictures

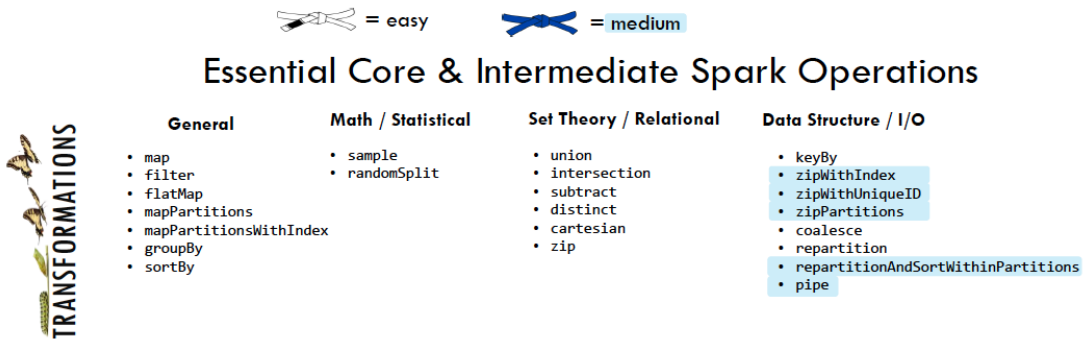
There are two main types of Spark operations: Transformations and Actions.

Note: Some people defined three types of operations: Transformations, Actions and Shuffles.



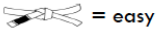
5.2.1 Spark Transformations

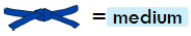
Transformations construct a new RDD from a previous one. For example, one common transformation is filtering data that matches a predicate.



5.2.2 Spark Actions

Actions, on the other hand, compute a result based on an RDD, and either return it to the driver program or save it to an external storage system (e.g., HDFS).





Essential Core & Intermediate PairRDD Operations

General

- flatMapValues
- groupByKey
- reduceByKey
- reduceByKeyLocally
- foldByKey
- aggregateByKey
- sortByKey
- combineByKey

Math / Statistical

- sampleByKey

Set Theory / Relational

- cogroup (=groupWith)
- join
- subtractByKey
- fullOuterJoin
- leftOuterJoin
- rightOuterJoin

Data Structure

- partitionBy

- reduce
- collect
- aggregate
- fold
- first
- take
- forEach
- top
- treeAggregate
- treeReduce
- forEachPartition
- collectAsMap

- count
- takeSample
- max
- min
- sum
- histogram
- mean
- variance
- stdev
- sampleVariance
- countApprox
- countApproxDistinct

- takeOrdered

- saveAsTextFile
- saveAsSequenceFile
- saveAsObjectFile
- saveAsHadoopDataset
- saveAsHadoopFile
- saveAsNewAPIHadoopDataset
- saveAsNewAPIHadoopFile

- keys
- values

- countByKey
- countByValue
- countByValueApprox
- countApproxDistinctByKey
- countApproxDistinctByKey
- countByKeyApprox
- sampleByKeyExact

CHAPTER

SIX

STATISTICS PRELIMINARY

Note: If you only know yourself, but not your opponent, you may win or may lose. If you know neither yourself nor your enemy, you will always endanger yourself – idiom, from Sunzi's Art of War

6.1 Notations

- m : the number of the samples
- n : the number of the features
- y_i : i-th label
- $\bar{y} = \frac{1}{m}\sum_{i=1}^{n} y_i$: the mean of y.

6.2 Measurement Formula

- Mean squared error

In statistics, the **MSE** (Mean Squared Error) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors or deviations—that is, the difference between the estimator and what is estimated.

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

- Root Mean squared error

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2}$$

- Total sum of squares

In statistical data analysis the **TSS** (Total Sum of Squares) is a quantity that appears as part of a standard way of presenting results of such analyses. It is defined as being the sum, over all observations, of the squared differences of each observation from the overall mean.

$$\text{TSS} = \sum_{i=1}^{m}(y_i - \bar{y})^2$$

- Residual Sum of Squares

$$\text{RSS} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

- Coefficient of determination R^2

$$R^2 := 1 - \frac{\text{RSS}}{\text{TSS}}.$$

6.3 Statistical Tests

6.3.1 Correlational Test

- Pearson correlation: Tests for the strength of the association between two continuous variables.
- Spearman correlation: Tests for the strength of the association between two ordinal variables (does not rely on the assumption of normal distributed data).
- Chi-square: Tests for the strength of the association between two categorical variables.

6.3.2 Comparison of Means test

- Paired T-test: Tests for difference between two related variables.
- Independent T-test: Tests for difference between two independent variables.
- ANOVA: Tests the difference between group means after any other variance in the outcome variable is accounted for.

6.3.3 Non-parametric Test

- Wilcoxon rank-sum test: Tests for difference between two independent variables - takes into account magnitude and direction of difference.
- Wilcoxon sign-rank test: Tests for difference between two related variables - takes into account magnitude and direction of difference.
- Sign test: Tests if two related variables are different – ignores magnitude of change, only takes into account direction.

CHAPTER

SEVEN

DATA EXPLORATION

Note: A journey of a thousand miles begins with a single step – idiom, from Laozi.

I wouldn't say that understanding your dataset is the most difficult thing in data science, but it is really important and time-consuming. Data Exploration is about describing the data by means of statistical and visualization techniques. We explore data in order to understand the features and bring important features to our models.

7.1 Univariate Analysis

In mathematics, univariate refers to an expression, equation, function or polynomial of only one variable. "Uni" means "one", so in other words your data has only one variable. So you do not need to deal with the causes or relationships in this step. Univariate analysis takes data, summarizes that variables (attributes) one by one and finds patterns in the data.

There are many ways that can describe patterns found in univariate data include central tendency (mean, mode and median) and dispersion: range, variance, maximum, minimum, quartiles (including the interquartile range), coefficient of variation and standard deviation. You also have several options for visualizing and describing data with univariate data. Such as `frequency Distribution Tables`, `bar Charts`, `histograms`, `frequency Polygons`, `pie Charts`.

The variable could be either categorical or numerical, I will demostrate the different statistical and visulization techniques to investigate each type of the variable.

- The Jupyter notebook can be download from Data Exploration.
- The data can be downloaf from German Credit.

7.1.1 Numerical Variables

- Describe

The `desctibe` function in `pandas` and `spark` will give us most of the statistical results, such as `min`, `median`, `max`, `quartiles` and `standard deviation`. With the help of the user defined function, you can get even more statistical results.

```
# selected varables for the demonstration
num_cols = ['Account Balance','No of dependents']
df.select(num_cols).describe().show()
```

```
+-------+------------------+-------------------+
|summary|   Account Balance|   No of dependents|
+-------+------------------+-------------------+
|  count|              1000|               1000|
|   mean|             2.577|              1.155|
| stddev|1.2576377271108936|0.36208577175319395|
|    min|                 1|                  1|
|    max|                 4|                  2|
+-------+------------------+-------------------+
```

You may find out that the default function in PySpark does not include the quartiles. The following function will help you to get the same results in Pandas

```
def describe_pd(df_in, columns, style):
    '''
    Function to union the basic stats results and deciles
    :param df_in: the input dataframe
    :param columns: the cloumn name list of the numerical variable
    :param style: the display style

    :return : the numerical describe info. of the input dataframe

    :author: MIng Chen and Wenqiang Feng
    :email:  von198@gmail.com
    '''

    if style == 1:
        percentiles = [25, 50, 75]
    else:
        percentiles = np.array(range(0, 110, 10))

    percs = np.transpose([np.percentile(df_in.select(x).collect(), percentiles) for x in
    percs = pd.DataFrame(percs, columns=columns)
    percs['summary'] = [str(p) + '%' for p in percentiles]

    spark_describe = df_in.describe().toPandas()
    new_df = pd.concat([spark_describe, percs],ignore_index=True)
    new_df = new_df.round(2)
    return new_df[['summary'] + columns]

describe_pd(df,num_cols,1)
```

```
+-------+------------------+-----------------+
|summary|   Account Balance| No of dependents|
+-------+------------------+-----------------+
|  count|            1000.0|           1000.0|
|   mean|             2.577|            1.155|
| stddev|1.2576377271108936|0.362085771753194|
|    min|               1.0|              1.0|
|    max|               4.0|              2.0|
|    25%|               1.0|              1.0|
|    50%|               2.0|              1.0|
|    75%|               4.0|              1.0|
+-------+------------------+-----------------+
```

Sometimes, because of the confidential data issues, you can not deliver the real data and your clients may ask more statistical results, such as `deciles`. You can apply the follwing function to achieve it.

```
describe_pd(df,num_cols,2)
```

```
+-------+------------------+-----------------+
|summary|   Account Balance| No of dependents|
+-------+------------------+-----------------+
|  count|            1000.0|           1000.0|
|   mean|             2.577|            1.155|
| stddev|1.2576377271108936|0.362085771753194|
|    min|               1.0|              1.0|
|    max|               4.0|              2.0|
|     0%|               1.0|              1.0|
|    10%|               1.0|              1.0|
|    20%|               1.0|              1.0|
|    30%|               2.0|              1.0|
|    40%|               2.0|              1.0|
|    50%|               2.0|              1.0|
|    60%|               3.0|              1.0|
|    70%|               4.0|              1.0|
|    80%|               4.0|              1.0|
|    90%|               4.0|              2.0|
|   100%|               4.0|              2.0|
+-------+------------------+-----------------+
```

- Skewness and Kurtosis

  This subsection comes from Wikipedia Skewness.

  In probability theory and statistics, skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The skewness value can be positive or negative, or undefined.For a unimodal distribution, negative skew commonly indicates that the tail is on the left side of the distribution, and positive skew indicates that the tail is on the right.

  Consider the two distributions in the figure just below. Within each graph, the values on the right side of the distribution taper differently from the values on the left side. These tapering sides are called tails, and they provide a visual means to determine which of the two kinds of skewness a distribution has:

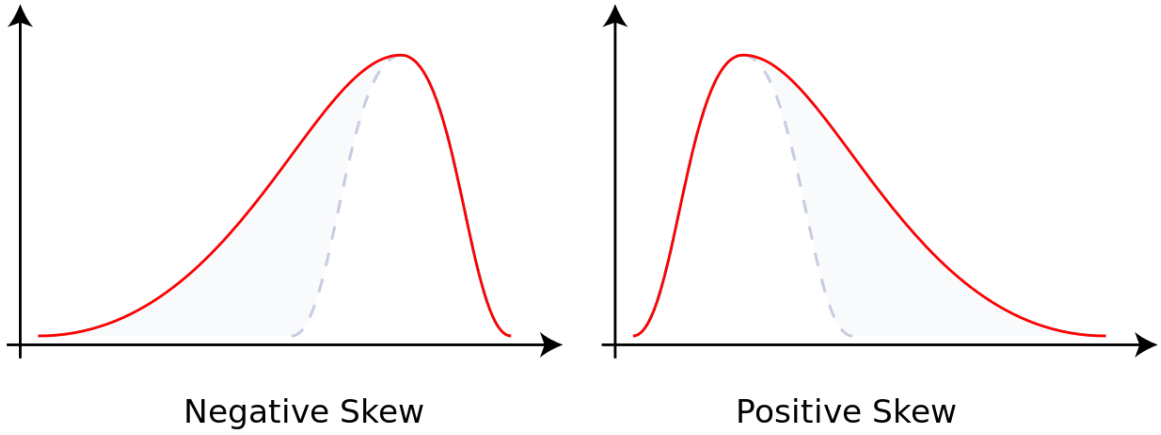
  1. negative skew: The left tail is longer; the mass of the distribution is concentrated on the right of the figure. The distribution is said to be left-skewed, left-tailed, or skewed to the left, despite the fact that the curve itself appears to be skewed or leaning to the right; left instead refers to the left tail being drawn out and, often, the mean being skewed to the left of a typical center of the data. A left-skewed distribution usually appears as a right-leaning curve.

  2. positive skew: The right tail is longer; the mass of the distribution is concentrated on the left of the figure. The distribution is said to be right-skewed, right-tailed, or skewed to the right, despite the fact that the curve itself appears to be skewed or leaning to the left; right instead refers to the right tail being drawn out and, often, the mean being skewed to the right of a typical center of the data. A right-skewed distribution usually appears as a left-leaning curve.

  This subsection comes from Wikipedia Kurtosis.

  In probability theory and statistics, kurtosis (kyrtos or kurtos, meaning "curved, arching") is a measure of the "tailedness" of the probability distribution of a real-valued random variable. In a similar way to the concept of skewness, kurtosis is a descriptor of the shape of a probability distribution and, just as for skewness, there are different ways of quantifying it for a theoretical

distribution and corresponding ways of estimating it from a sample from a population.




```
from pyspark.sql.functions import col, skewness, kurtosis
df.select(skewness(var),kurtosis(var)).show()
```

```
+---------------------+---------------------+
|skewness(Age (years))|kurtosis(Age (years))|
+---------------------+---------------------+
|   1.0231743160548064|   0.6114371688367672|
+---------------------+---------------------+
```

Warning: Sometimes the statistics can be misleading!

F. J. Anscombe once said that make both calculations and graphs. Both sorts of output should be studied; each will contribute to understanding. These 13 datasets in Figure *Same Stats, Different Graphs* (the Datasaurus, plus 12 others) each have the same summary statistics (x/y mean, x/y standard deviation, and Pearson's correlation) to two decimal places, while being drastically different in appearance. This work describes the technique we developed to create this dataset, and others like it. More details and interesting results can be found in Same Stats Different Graphs.

- Histogram

Warning: Histograms are often confused with Bar graphs!

The fundamental difference between histogram and bar graph will help you to identify the two easily is that there are gaps between bars in a bar graph but in the histogram, the bars are adjacent to each other. The interested reader is referred to Difference Between Histogram and Bar Graph.

```
var = 'Age (years)'
x = data1[var]
bins = np.arange(0, 100, 5.0)

plt.figure(figsize=(10,8))
# the histogram of the data
plt.hist(x, bins, alpha=0.8, histtype='bar', color='gold',
         ec='black',weights=np.zeros_like(x) + 100. / x.size)

plt.xlabel(var)
plt.ylabel('percentage')
plt.xticks(bins)
plt.show()
```

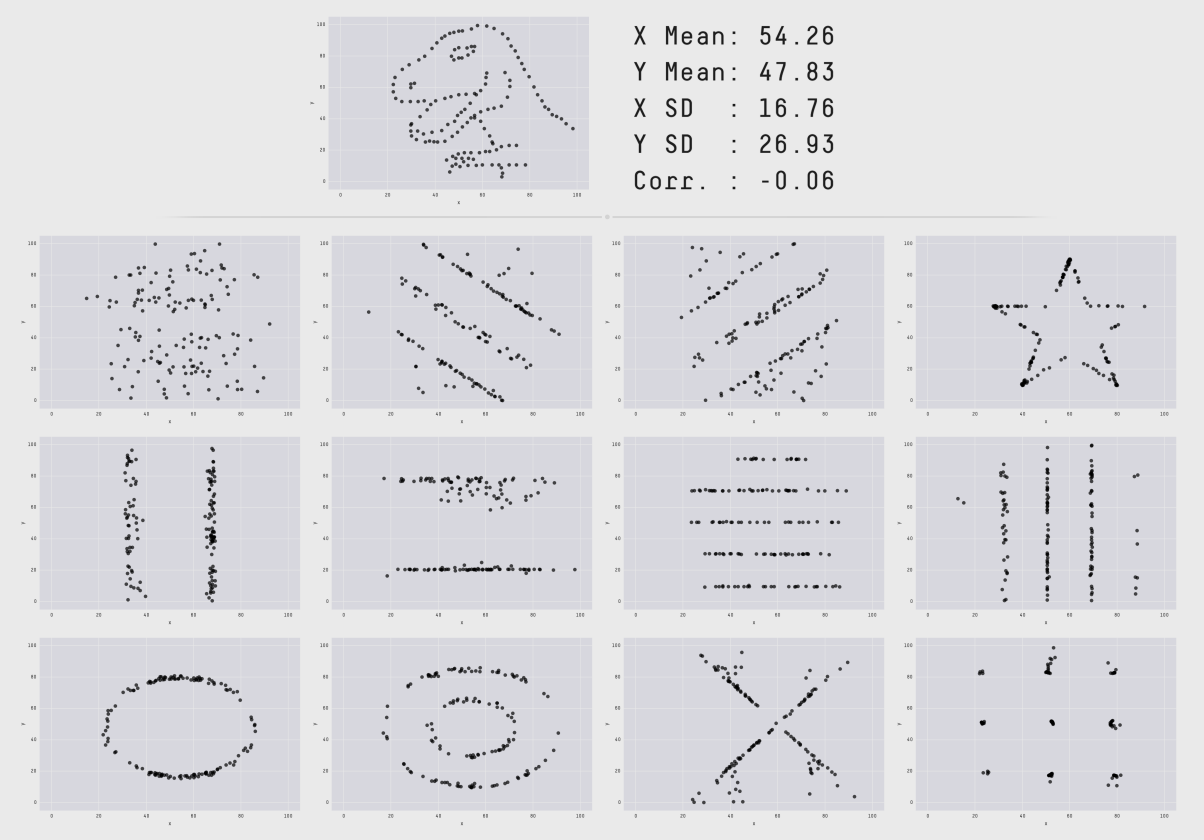
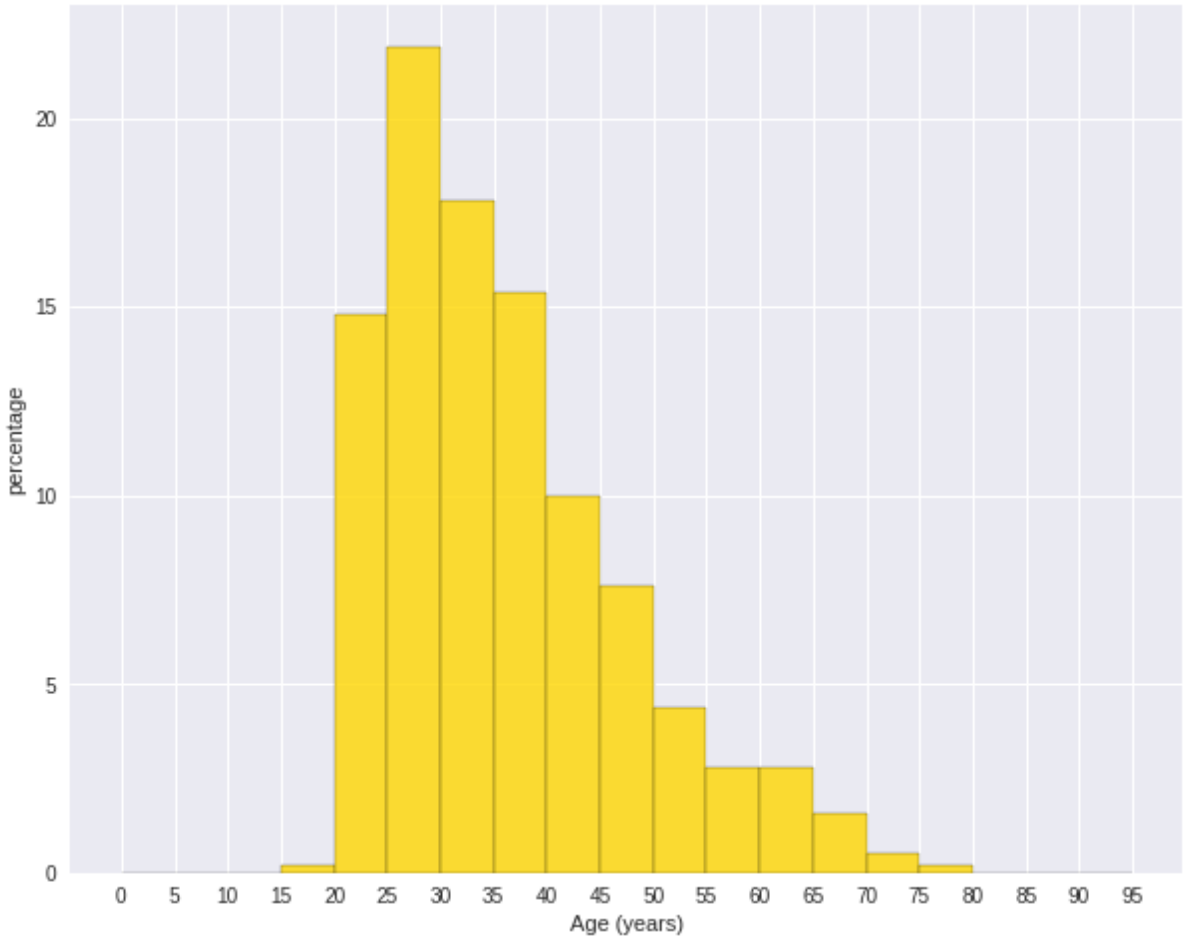
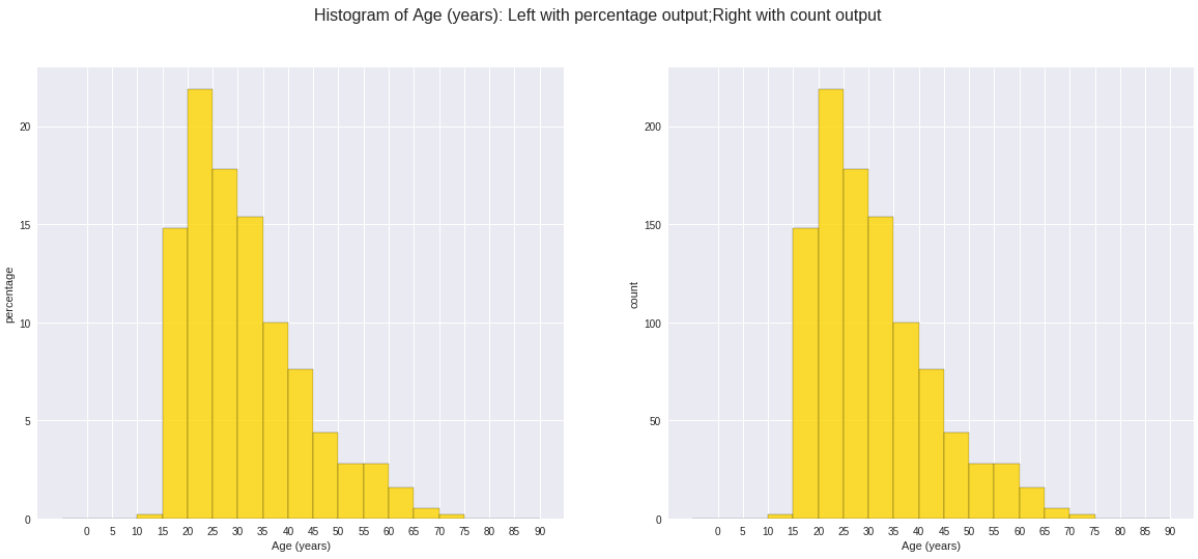


Figure 7.1: Same Stats, Different Graphs

```
fig.savefig(var+".pdf", bbox_inches='tight')

var = 'Age (years)'
x = data1[var]
bins = np.arange(0, 100, 5.0)


################################################################################
hist, bin_edges = np.histogram(x,bins,
                               weights=np.zeros_like(x) + 100. / x.size)
# make the histogram

fig = plt.figure(figsize=(20, 8))
ax = fig.add_subplot(1, 2, 1)

# Plot the histogram heights against integers on the x axis
ax.bar(range(len(hist)),hist,width=1,alpha=0.8,ec ='black', color='gold')
# # Set the ticks to the middle of the bars
ax.set_xticks([0.5+i for i,j in enumerate(hist)])
# Set the xticklabels to a string that tells us what the bin edges were
labels =['{}'.format(int(bins[i+1])) for i,j in enumerate(hist)]
labels.insert(0,'0')
ax.set_xticklabels(labels)
plt.xlabel(var)
plt.ylabel('percentage')

################################################################################
```


20
15
10
5
0
percentage
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95
Age (years)

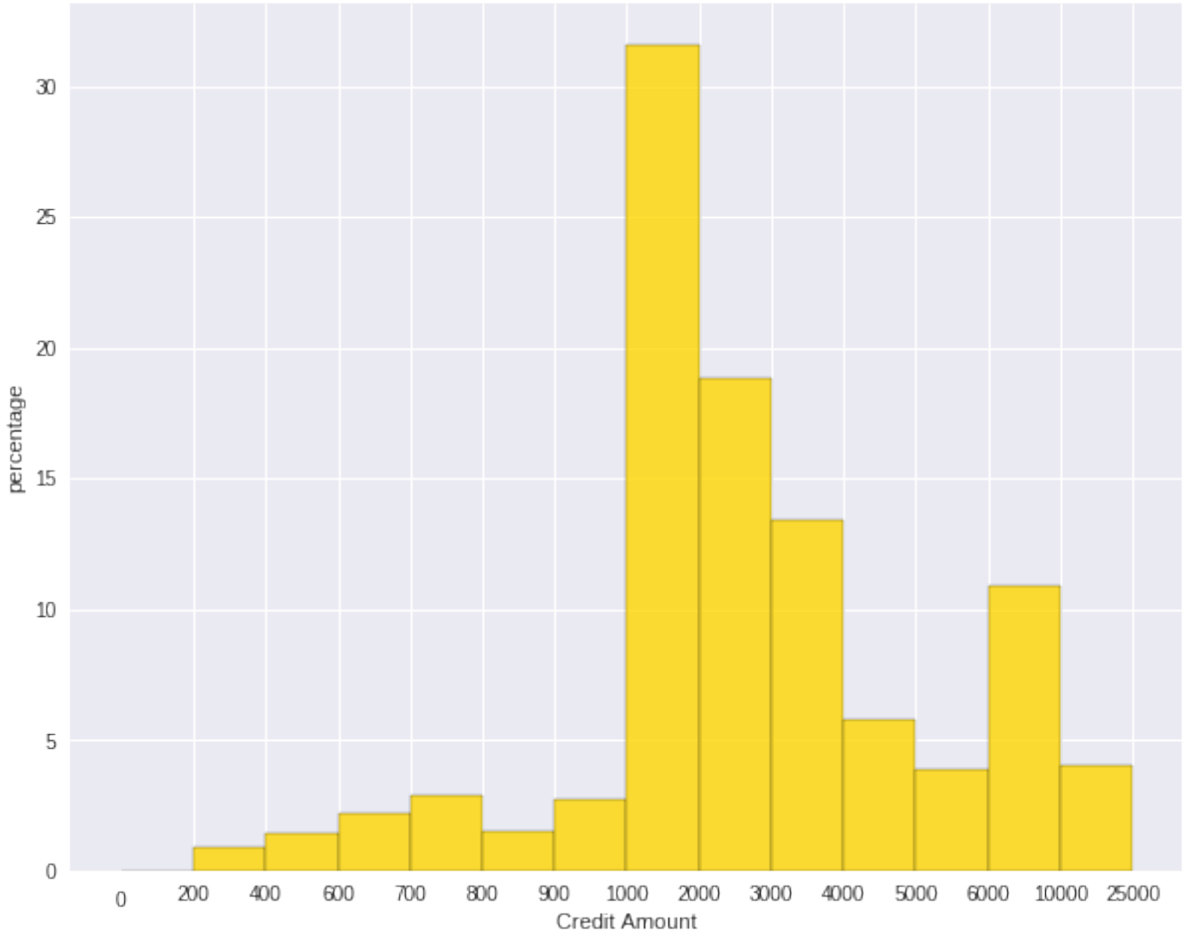
```
hist, bin_edges = np.histogram(x,bins) # make the histogram

ax = fig.add_subplot(1, 2, 2)
# Plot the histogram heights against integers on the x axis
ax.bar(range(len(hist)),hist,width=1,alpha=0.8,ec ='black', color='gold')

# # Set the ticks to the middle of the bars
ax.set_xticks([0.5+i for i,j in enumerate(hist)])

# Set the xticklabels to a string that tells us what the bin edges were
labels ={}.format(int(bins[i+1])) for i,j in enumerate(hist)]
labels.insert(0,'0')
ax.set_xticklabels(labels)
plt.xlabel(var)
plt.ylabel('count')
plt.suptitle('Histogram of {}: Left with percentage output;Right with count output'
             .format(var), size=16)
plt.show()

fig.savefig(var+".pdf", bbox_inches='tight')
```



Sometimes, some people will ask you to plot the unequal width (invalid argument for histogram) of the bars. YOu can still achieve it by the follwing trick.

```
var = 'Credit Amount'
plot_data = df.select(var).toPandas()
x= plot_data[var]

bins =[0,200,400,600,700,800,900,1000,2000,3000,4000,5000,6000,10000,25000]

hist, bin_edges = np.histogram(x,bins,weights=np.zeros_like(x) + 100. / x.size) # make t

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(1, 1, 1)
# Plot the histogram heights against integers on the x axis
ax.bar(range(len(hist)),hist,width=1,alpha=0.8,ec ='black',color = 'gold')

# # Set the ticks to the middle of the bars
```

```
ax.set_xticks([0.5+i for i,j in enumerate(hist)])

# Set the xticklabels to a string that tells us what the bin edges were
#labels =['{}k'.format(int(bins[i+1]/1000)) for i,j in enumerate(hist)]
labels =['{}'.format(bins[i+1]) for i,j in enumerate(hist)]
labels.insert(0,'0')
ax.set_xticklabels(labels)
#plt.text(-0.6, -1.4,'0')
plt.xlabel(var)
plt.ylabel('percentage')
plt.show()
```

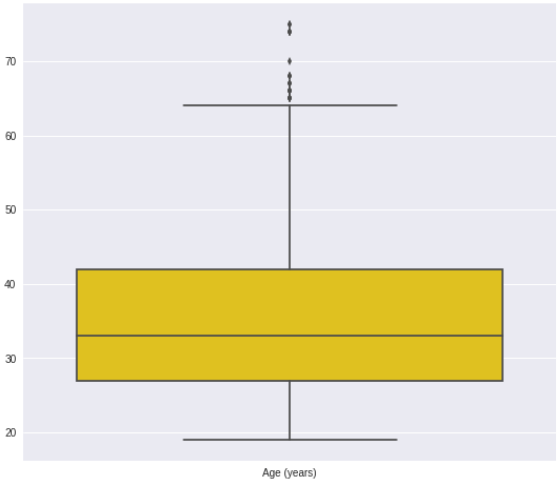


- Box plot and violin plot

Note that although violin plots are closely related to Tukey's (1977) box plots, the violin plot can show more information than box plot. When we perform an exploratory analysis, nothing about the samples could be known. So the distribution of the samples can not be assumed to a normal distribution and usually when you get a big data, the normal distribution will show some out liars in box plot.

However, the violin plots are potentially misleading for smaller sample sizes, where the density plots can appear to show interesting features (and group-differences therein) even when produced for standard normal data. Some poster suggested the sample size should larger that 250. The sample sizes (e.g. n>250 or ideally even larger), where the kernel density plots provide a reasonably accurate representation of the distributions, potentially showing nuances such as bimodality or other forms of non-normality that would be invisible or less clear in box plots. More details can be found in A simple comparison of box plots and violin plots.

```
x = df.select(var).toPandas()

fig = plt.figure(figsize=(20, 8))
ax = fig.add_subplot(1, 2, 1)
ax = sns.boxplot(data=x)

ax = fig.add_subplot(1, 2, 2)
ax = sns.violinplot(data=x)
```



7.1.2 Categorical Variables

Compared with the numerical variables, the categorical variables are much more easier to do the exploration.

- Frequency table

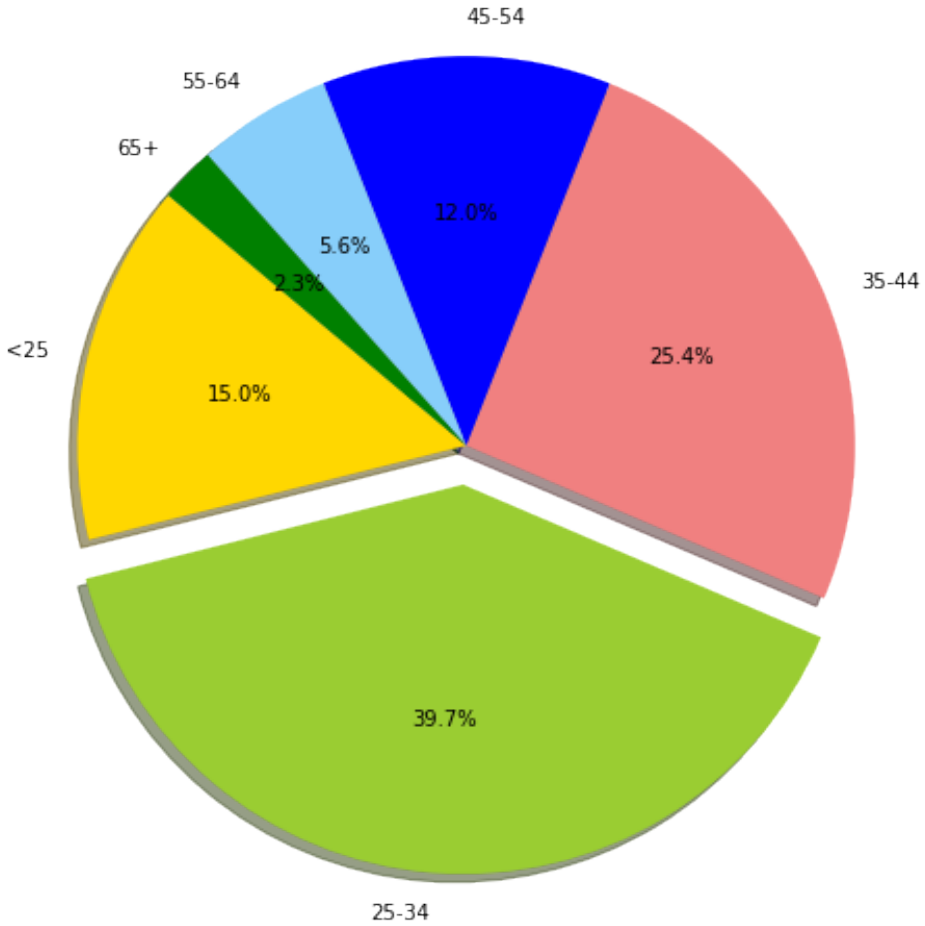
```
from pyspark.sql import functions as F
from pyspark.sql.functions import rank,sum,col
from pyspark.sql import Window

window = Window.rowsBetween(Window.unboundedPreceding,Window.unboundedFollowing)
# withColumn('Percent %',F.format_string("%5.0f%%\n",col('Credit_num')*100/col('total'))
tab = df.select(['age_class','Credit Amount']).\
   groupBy('age_class').\
   agg(F.count('Credit Amount').alias('Credit_num'),
       F.mean('Credit Amount').alias('Credit_avg'),
       F.min('Credit Amount').alias('Credit_min'),
       F.max('Credit Amount').alias('Credit_max')).\
   withColumn('total',sum(col('Credit_num')).over(window)).\
   withColumn('Percent',col('Credit_num')*100/col('total')).\
   drop(col('total'))
```

```
+---------+----------+------------------+----------+----------+-------+
|age_class|Credit_num|        Credit_avg|Credit_min|Credit_max|Percent|
+---------+----------+------------------+----------+----------+-------+
|    45-54|       120|3183.0666666666666|       338|     12612|   12.0|
|      <25|       150| 2970.733333333333|       276|     15672|   15.0|
|    55-64|        56| 3493.660714285714|       385|     15945|    5.6|
|    35-44|       254| 3403.771653543307|       250|     15857|   25.4|
|    25-34|       397| 3298.823677581864|       343|     18424|   39.7|
```

```
|      65+|         23|3210.1739130434785|       571|     14896|    2.3|
+---------+-----------+------------------+----------+----------+-------+
```

- Pie plot

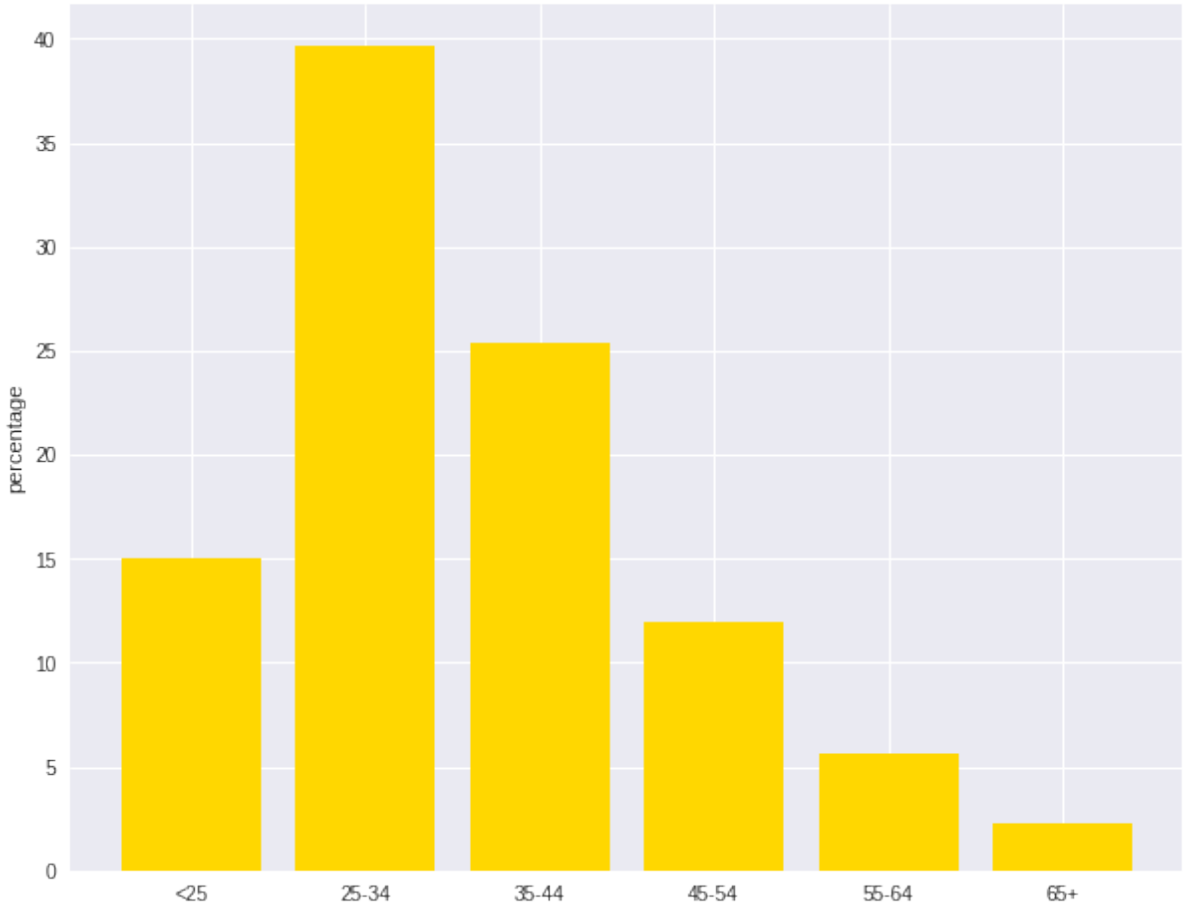
```
# Data to plot
labels = plot_data.age_class
sizes =  plot_data.Percent
colors = ['gold', 'yellowgreen', 'lightcoral','blue', 'lightskyblue','green','red']
explode = (0, 0.1, 0, 0,0,0)  # explode 1st slice

# Plot
plt.figure(figsize=(10,8))
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```



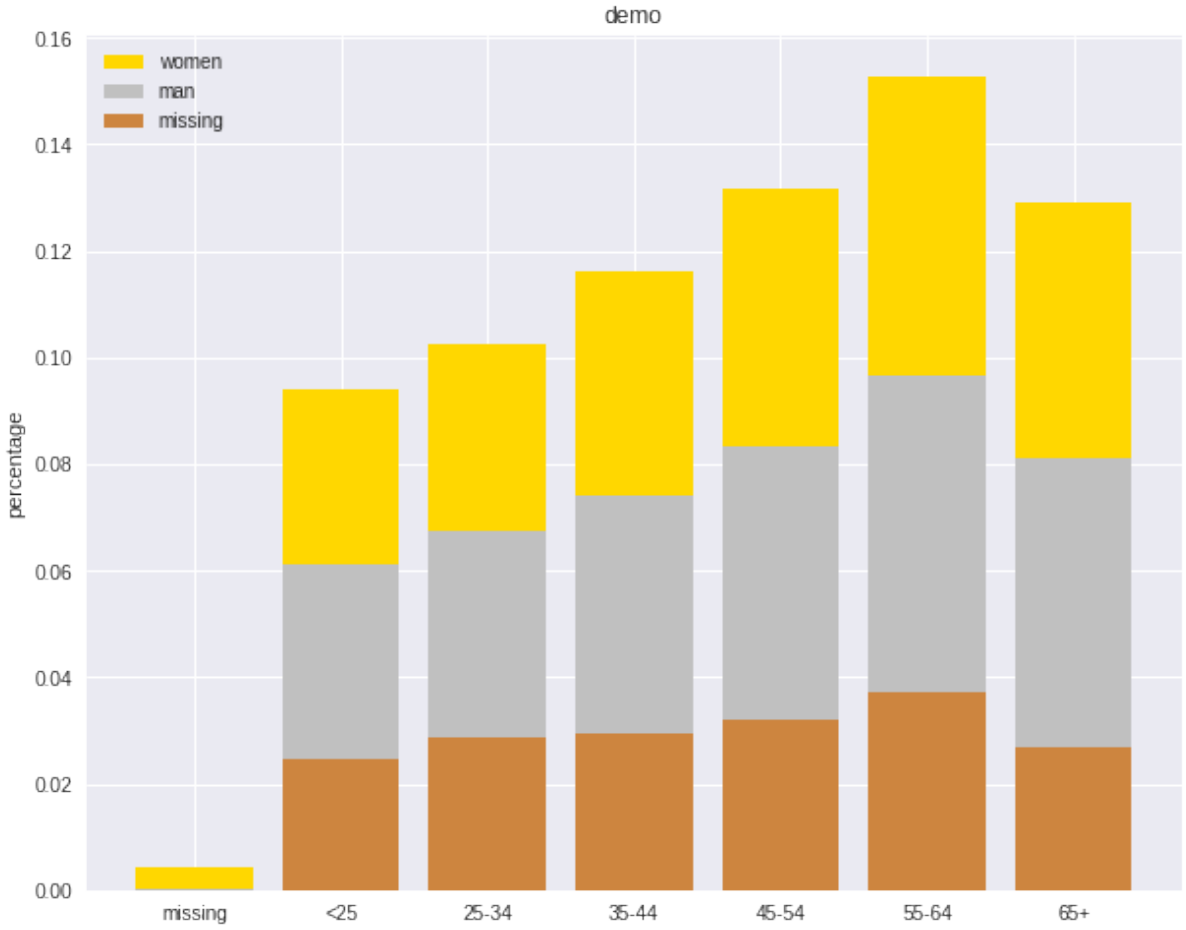
- Bar plot

```
labels = plot_data.age_class
missing = plot_data.Percent
ind = [x for x, _ in enumerate(labels)]

plt.figure(figsize=(10,8))
plt.bar(ind, missing, width=0.8, label='missing', color='gold')
```

```
plt.xticks(ind, labels)
plt.ylabel("percentage")

plt.show()
```




```
labels = ['missing', '<25', '25-34', '35-44', '45-54','55-64','65+']
missing = np.array([0.000095, 0.024830, 0.028665, 0.029477, 0.031918,0.037073,0.026699])
man = np.array([0.000147, 0.036311, 0.038684, 0.044761, 0.051269, 0.059542, 0.054259])
women = np.array([0.004035, 0.032935, 0.035351, 0.041778, 0.048437, 0.056236,0.048091])
ind = [x for x, _ in enumerate(labels)]

plt.figure(figsize=(10,8))
plt.bar(ind, women, width=0.8, label='women', color='gold', bottom=man+missing)
plt.bar(ind, man, width=0.8, label='man', color='silver', bottom=missing)
plt.bar(ind, missing, width=0.8, label='missing', color='#CD853F')

plt.xticks(ind, labels)
plt.ylabel("percentage")
plt.legend(loc="upper left")
plt.title("demo")

plt.show()
```


demo
women
man
missing
percentage
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
missing
<25
25-34
35-44
45-54
55-64
65+

7.2 Multivariate Analysis

In this section, I will only demostrate the bivariate analysis. Since the multivariate analysis is the generation of the bivariate.

7.2.1 Numerical V.S. Numerical

- Correlation matrix

```
from pyspark.mllib.stat import Statistics
import pandas as pd

corr_data = df.select(num_cols)

col_names = corr_data.columns
features = corr_data.rdd.map(lambda row: row[0:])
corr_mat=Statistics.corr(features, method="pearson")
corr_df = pd.DataFrame(corr_mat)
corr_df.index, corr_df.columns = col_names, col_names

print(corr_df.to_string())
```

```
+--------------------+--------------------+
|     Account Balance|    No of dependents|
+--------------------+--------------------+
|                 1.0|-0.01414542650320914|
|-0.01414542650320914|                 1.0|
+--------------------+--------------------+
```

- Scatter Plot

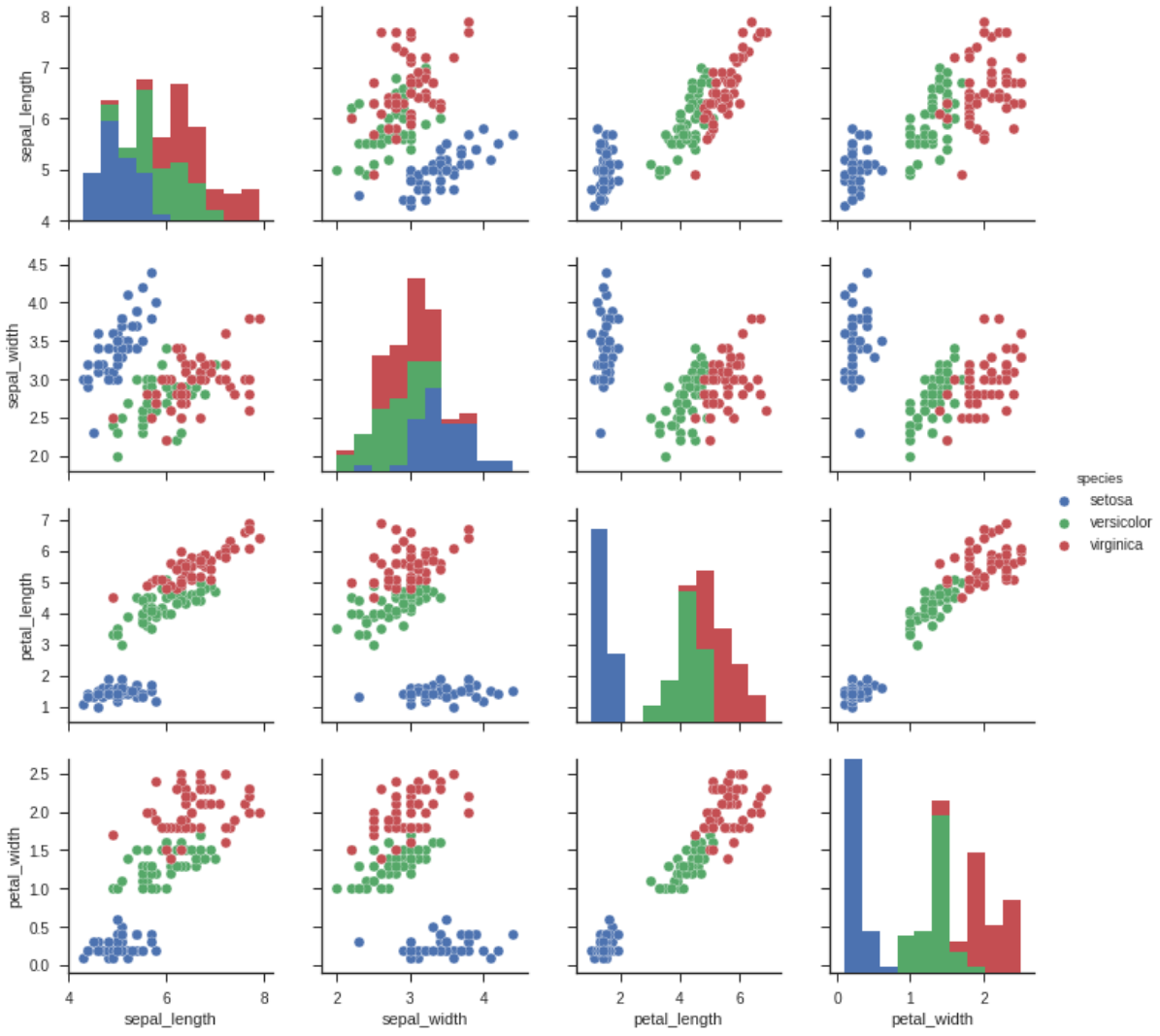
```
import seaborn as sns
sns.set(style="ticks")

df = sns.load_dataset("iris")
sns.pairplot(df, hue="species")
plt.show()
```

7.2.2 Categorical V.S. Categorical

- Pearson's Chi-squared test

> **Warning:** `pyspark.ml.stat` is only available in Spark 2.4.0.

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import ChiSquareTest

data = [(0.0, Vectors.dense(0.5, 10.0)),
        (0.0, Vectors.dense(1.5, 20.0)),
        (1.0, Vectors.dense(1.5, 30.0)),
        (0.0, Vectors.dense(3.5, 30.0)),
        (0.0, Vectors.dense(3.5, 40.0)),
        (1.0, Vectors.dense(3.5, 40.0))]
df = spark.createDataFrame(data, ["label", "features"])
```


sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica

```
r = ChiSquareTest.test(df, "features", "label").head()
print("pValues: " + str(r.pValues))
print("degreesOfFreedom: " + str(r.degreesOfFreedom))
print("statistics: " + str(r.statistics))

pValues: [0.687289278791,0.682270330336]
degreesOfFreedom: [2, 3]
statistics: [0.75,1.5]
```

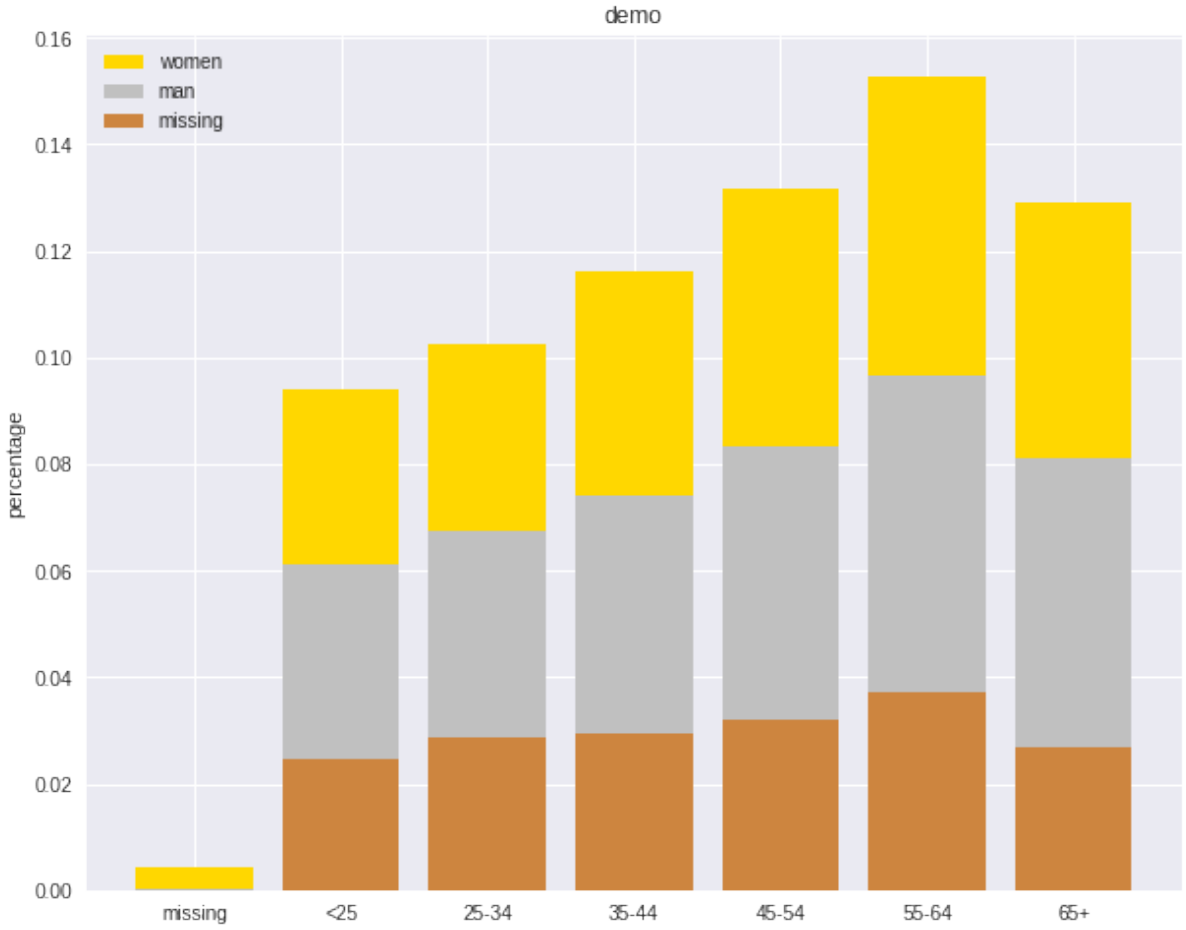
- Cross table

```
df.stat.crosstab("age_class", "Occupation").show()

+--------------------+---+---+---+---+
|age_class_Occupation|  1|  2|  3|  4|
+--------------------+---+---+---+---+
|                 <25|  4| 34|108|  4|
|               55-64|  1| 15| 31|  9|
|               25-34|  7| 61|269| 60|
|               35-44|  4| 58|143| 49|
|                 65+|  5|  3|  6|  9|
|               45-54|  1| 29| 73| 17|
+--------------------+---+---+---+---+
```

- Stacked plot

```
labels = ['missing', '<25', '25-34', '35-44', '45-54','55-64','65+']
missing = np.array([0.000095, 0.024830, 0.028665, 0.029477, 0.031918,0.037073,0.026699])
man = np.array([0.000147, 0.036311, 0.038684, 0.044761, 0.051269, 0.059542, 0.054259])
women = np.array([0.004035, 0.032935, 0.035351, 0.041778, 0.048437, 0.056236,0.048091])
ind = [x for x, _ in enumerate(labels)]

plt.figure(figsize=(10,8))
plt.bar(ind, women, width=0.8, label='women', color='gold', bottom=man+missing)
plt.bar(ind, man, width=0.8, label='man', color='silver', bottom=missing)
plt.bar(ind, missing, width=0.8, label='missing', color='#CD853F')

plt.xticks(ind, labels)
plt.ylabel("percentage")
plt.legend(loc="upper left")
plt.title("demo")

plt.show()
```

7.2.3 Numerical V.S. Categorical


demo
women
man
missing
percentage
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
missing
<25
25-34
35-44
45-54
55-64
65+

CHAPTER

EIGHT

REGRESSION

Note: A journey of a thousand miles begins with a single step – old Chinese proverb

In statistical modeling, regression analysis focuses on investigating the relationship between a dependent variable and one or more independent variables. Wikipedia Regression analysis

In data mining, Regression is a model to represent the relationship between the value of lable (or target, it is numerical variable) and on one or more features (or predictors they can be numerical and categorical variables).

8.1 Linear Regression

8.1.1 Introduction

Given that a data set $\{ x_{i1}, \ldots, x_{in}, y_i\}_{i=1}^{m}$ which contains n features (variables) and m samples (data points), in simple linear regression model for modeling m data points with one independent variable: x_{i1}, the formula is given by:

$$y_i = \beta_0 + \beta_1 x_{i1}, \text{where}, i = 1, \cdots m.$$

In matrix notation, the data set is written as $\mathbf{X} = [\mathbf{X}_1, \cdots, \mathbf{X}_n]$ with $\mathbf{X}_i = \{x_{.i}\}_{i=1}^{n}$, $\mathbf{y} = \{y_i\}_{i=1}^{m}$ and $\boldsymbol{\beta}^\top = \{\beta_i\}_{i=1}^{m}$. Then the normal equations are written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}.$$

8.1.2 How to solve it?

1. Direct Methods (For more information please refer to my Prelim Notes for Numerical Analysis)
    - For squared or rectangular matrices
        - Singular Value Decomposition
        - Gram-Schmidt orthogonalization
        - QR Decomposition
    - For squared matrices

    - LU Decomposition
    - Cholesky Decomposition
    - Regular Splittings

2. Iterative Methods
    - Stationary cases iterative method
        - Jacobi Method
        - Gauss-Seidel Method
        - Richardson Method
        - Successive Over Relaxation (SOR) Method
    - Dynamic cases iterative method
        - Chebyshev iterative Method
        - Minimal residuals Method
        - Minimal correction iterative method
        - Steepest Descent Method
        - Conjugate Gradients Method

8.1.3 Demo

- The Jupyter notebook can be download from Linear Regression which was implemented without using Pipeline.
- The Jupyter notebook can be download from Linear Regression with Pipeline which was implemented with using Pipeline.
- I will only present the code with pipeline style in the following.
- For more details about the parameters, please visit Linear Regression API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark regression example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                        options(header='true', \
                        inferschema='true').\
            load("../data/Advertising.csv",header=True);
```

check the data set

```
df.show(5,True)
df.printSchema()
```

Then you will get

```
+-----+-----+---------+-----+
|   TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
| 17.2| 45.9|     69.3|  9.3|
|151.5| 41.3|     58.5| 18.5|
|180.8| 10.8|     58.4| 12.9|
+-----+-----+---------+-----+
only showing top 5 rows

root
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

You can also get the Statistical resutls from the data frame (Unfortunately, it only works for numerical).

```
df.describe().show()
```

Then you will get

```
+-------+-----------------+------------------+------------------+------------------+
|summary|               TV|             Radio|         Newspaper|             Sales|
+-------+-----------------+------------------+------------------+------------------+
|  count|              200|               200|               200|               200|
|   mean|         147.0425|23.264000000000024|30.553999999999995|14.022500000000003|
| stddev|85.85423631490805|14.846809176168728| 21.77862083852283| 5.217456565710477|
|    min|              0.7|               0.0|               0.3|               1.6|
|    max|            296.4|              49.6|             114.0|              27.0|
+-------+-----------------+------------------+------------------+------------------+
```

3. Convert the data to dense vector (**features** and **label**)

```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# I provide two ways to build the features and labels

# method 1 (good for small feature):
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
#                                       row["Radio"],
#                                       row["Newspaper"]]))

# Method 2 (good for large features):
def transData(data):
return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label'])
```

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Figure 8.1: Sales distribution

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for unsupervised learn
    :param df: the dataframe
```

```
        :param categoricalCols: the name list of the categorical data
        :param continuousCols:  the name list of the numerical data
        :return k: feature matrix

        :author: Wenqiang Feng
        :email:  von198@gmail.com
        '''

        indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                     for c in categoricalCols ]

        # default setting: dropLast=True
        encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                     outputCol="{0}_encoded".format(indexer.getOutputCol()))
                     for indexer in indexers ]

        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
```

4. Transform the dataset to DataFrame

```
transformed= transData(df)
transformed.show(5)
```

```
+-----------------+-----+
|         features|label|
+-----------------+-----+
|[230.1,37.8,69.2]| 22.1|
| [44.5,39.3,45.1]| 10.4|
| [17.2,45.9,69.3]|  9.3|
|[151.5,41.3,58.5]| 18.5|
|[180.8,10.8,58.4]| 12.9|
+-----------------+-----+
only showing top 5 rows
```

Note: You will find out that all of the supervised machine learning algorithms in Spark are based on the **features** and **label** (unsupervised machine learning algorithms in Spark are based on the **features**). That is to say, you can play with all of the machine learning algorithms in Spark when you get ready the **features** and **label** in pipeline architecture.

5. Deal With Categorical Variables

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4 distinct values are treated as continuou
```

```
featureIndexer = VectorIndexer(inputCol="features", \
                               outputCol="indexedFeatures",\
                               maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
```

Now you check your dataset with

```
data.show(5,True)
```

you will get

```
+-----------------+-----+-----------------+
|         features|label|  indexedFeatures|
+-----------------+-----+-----------------+
|[230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2]|
| [44.5,39.3,45.1]| 10.4| [44.5,39.3,45.1]|
| [17.2,45.9,69.3]|  9.3| [17.2,45.9,69.3]|
|[151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5]|
|[180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4]|
+-----------------+-----+-----------------+
only showing top 5 rows
```

6. Split the data into training and test sets (40% held out for testing)

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])
```

You can check your train and test data as follows (In my opinion, it is always to good to keep tracking your data during prototype pahse):

```
trainingData.show(5)
testData.show(5)
```

Then you will get

```
+---------------+-----+---------------+
|       features|label|indexedFeatures|
+---------------+-----+---------------+
| [4.1,11.6,5.7]|  3.2| [4.1,11.6,5.7]|
| [5.4,29.9,9.4]|  5.3| [5.4,29.9,9.4]|
|[7.3,28.1,41.4]|  5.5|[7.3,28.1,41.4]|
|[7.8,38.9,50.6]|  6.6|[7.8,38.9,50.6]|
|  [8.6,2.1,1.0]|  4.8|  [8.6,2.1,1.0]|
+---------------+-----+---------------+
only showing top 5 rows

+----------------+-----+----------------+
|        features|label| indexedFeatures|
+----------------+-----+----------------+
|  [0.7,39.6,8.7]|  1.6|  [0.7,39.6,8.7]|
|  [8.4,27.2,2.1]|  5.7|  [8.4,27.2,2.1]|
|[11.7,36.9,45.2]|  7.3|[11.7,36.9,45.2]|
|[13.2,15.9,49.6]|  5.6|[13.2,15.9,49.6]|
|[16.9,43.7,89.4]|  8.7|[16.9,43.7,89.4]|
+----------------+-----+----------------+
only showing top 5 rows
```

7. Fit Ordinary Least Square Regression Model

For more details about the parameters, please visit Linear Regression API .

```
# Import LinearRegression class
from pyspark.ml.regression import LinearRegression

# Define LinearRegression algorithm
lr = LinearRegression()
```

8. Pipeline Architecture

```
# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, lr])

model = pipeline.fit(trainingData)
```

9. Summary of the Model

Spark has a poor summary function for data and model. I wrote a summary function which has similar format as **R** output for the linear regression in PySpark.

```
def modelsummary(model):
    import numpy as np
    print ("Note: the last rows are the information for Intercept")
    print ("##","------------------------------------------------")
    print ("##","  Estimate   |   Std.Error | t Values  |  P-value")
    coef = np.append(list(model.coefficients),model.intercept)
    Summary=model.summary

    for i in range(len(Summary.pValues)):
        print ("##",'{:10.6f}'.format(coef[i]),\
        '{:10.6f}'.format(Summary.coefficientStandardErrors[i]),\
        '{:8.3f}'.format(Summary.tValues[i]),\
        '{:10.6f}'.format(Summary.pValues[i]))

    print ("##",'---')
    print ("##","Mean squared error: % .6f" \
           % Summary.meanSquaredError, ", RMSE: % .6f" \
           % Summary.rootMeanSquaredError )
    print ("##","Multiple R-squared: %f" % Summary.r2, ", \
            Total iterations: %i"% Summary.totalIterations)

modelsummary(model.stages[-1])
```

You will get the following summary results:

```
Note: the last rows are the information for Intercept
('##', '------------------------------------------------')
('##', '  Estimate   |   Std.Error | t Values  |  P-value')
('##', '  0.044186', '  0.001663', '  26.573', '  0.000000')
('##', '  0.206311', '  0.010846', '  19.022', '  0.000000')
('##', '  0.001963', '  0.007467', '   0.263', '  0.793113')
('##', '  2.596154', '  0.379550', '   6.840', '  0.000000')
('##', '---')
('##', 'Mean squared error:  2.588230', ', RMSE:  1.608798')
('##', 'Multiple R-squared: 0.911869', ',            Total iterations: 1')
```

10. Make predictions

```
# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
+----------------+-----+------------------+
|        features|label|        prediction|
+----------------+-----+------------------+
|  [0.7,39.6,8.7]|  1.6| 10.81405928637388|
|  [8.4,27.2,2.1]|  5.7| 8.583086404079918|
|[11.7,36.9,45.2]|  7.3|10.814712818232422|
|[13.2,15.9,49.6]|  5.6| 6.557106943899219|
|[16.9,43.7,89.4]|  8.7|12.534151375058645|
+----------------+-----+------------------+
only showing top 5 rows
```

9. Evaluation

```
from pyspark.ml.evaluation import RegressionEvaluator
# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(labelCol="label",
                                predictionCol="prediction",
                                metricName="rmse")

rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

The final Root Mean Squared Error (RMSE) is as follows:

```
Root Mean Squared Error (RMSE) on test data = 1.63114
```

You can also check the R^2 value for the test data:

```
y_true = predictions.select("label").toPandas()
y_pred = predictions.select("prediction").toPandas()

import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {0}'.format(r2_score))
```

Then you will get

```
r2_score: 0.854486655585
```

Warning: You should know most softwares are using different formula to calculate the R^2 value when no intercept is included in the model. You can get more information from the disscussion at StackExchange.

8.2 Generalized linear regression

8.2.1 Introduction

8.2.2 How to solve it?

8.2.3 Demo

- The Jupyter notebook can be download from Generalized Linear Regression.
- For more details about the parameters, please visit Generalized Linear Regression API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark regression example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                       options(header='true', \
                       inferschema='true').\
            load("../data/Advertising.csv",header=True);
```

check the data set

```
df.show(5,True)
df.printSchema()
```

Then you will get

```
+-----+-----+---------+-----+
|   TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
| 17.2| 45.9|     69.3|  9.3|
|151.5| 41.3|     58.5| 18.5|
|180.8| 10.8|     58.4| 12.9|
+-----+-----+---------+-----+
only showing top 5 rows

root
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

You can also get the Statistical resutls from the data frame (Unfortunately, it only works for numerical).

```
df.describe().show()
```

Then you will get

```
+-------+-----------------+------------------+------------------+------------------+
|summary|               TV|             Radio|         Newspaper|             Sales|
+-------+-----------------+------------------+------------------+------------------+
|  count|              200|               200|               200|               200|
|   mean|         147.0425|23.264000000000024|30.553999999999995|14.022500000000003|
| stddev|85.85423631490805|14.846809176168728| 21.77862083852283| 5.217456565710477|
|    min|              0.7|               0.0|               0.3|               1.6|
|    max|            296.4|              49.6|             114.0|              27.0|
+-------+-----------------+------------------+------------------+------------------+
```

3. Convert the data to dense vector (**features** and **label**)

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for unsupervised learn
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
```

```
                    for c in categoricalCols ]

        # default setting: dropLast=True
        encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                     outputCol="{0}_encoded".format(indexer.getOutputCol()))
                     for indexer in indexers ]

        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
```

```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# I provide two ways to build the features and labels

# method 1 (good for small feature):
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
#                                       row["Radio"],
#                                       row["Newspaper"]]))

# Method 2 (good for large features):
def transData(data):
return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label'])

transformed= transData(df)
transformed.show(5)
```

```
+-----------------+-----+
|         features|label|
+-----------------+-----+
|[230.1,37.8,69.2]| 22.1|
| [44.5,39.3,45.1]| 10.4|
| [17.2,45.9,69.3]|  9.3|
|[151.5,41.3,58.5]| 18.5|
|[180.8,10.8,58.4]| 12.9|
+-----------------+-----+
only showing top 5 rows
```

Note: You will find out that all of the machine learning algorithms in Spark are based on the **features** and **label**. That is to say, you can play with all of the machine learning algorithms in Spark when you get ready the **features** and **label**.

4. Convert the data to dense vector

```
# convert the data to dense vector
def transData(data):
```

```
    return data.rdd.map(lambda r: [r[-1], Vectors.dense(r[:-1])]).\
           toDF(['label','features'])

from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

data= transData(df)
data.show()
```

5. Deal with the Categorical variables

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4
# distinct values are treated as continuous.

featureIndexer = VectorIndexer(inputCol="features", \
                               outputCol="indexedFeatures",\
                               maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
```

When you check you data at this point, you will get

```
+-----------------+-----+-----------------+
|         features|label|  indexedFeatures|
+-----------------+-----+-----------------+
|[230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2]|
| [44.5,39.3,45.1]| 10.4| [44.5,39.3,45.1]|
| [17.2,45.9,69.3]|  9.3| [17.2,45.9,69.3]|
|[151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5]|
|[180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4]|
+-----------------+-----+-----------------+
only showing top 5 rows
```

6. Split the data into training and test sets (40% held out for testing)

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])
```

You can check your train and test data as follows (In my opinion, it is always to good to keep tracking your data during prototype pahse):

```
trainingData.show(5)
testData.show(5)
```

Then you will get

```
+---------------+-----+---------------+
|       features|label| indexedFeatures|
+---------------+-----+---------------+
|  [5.4,29.9,9.4]|  5.3|  [5.4,29.9,9.4]|
| [7.8,38.9,50.6]|  6.6| [7.8,38.9,50.6]|
|  [8.4,27.2,2.1]|  5.7|  [8.4,27.2,2.1]|
| [8.7,48.9,75.0]|  7.2| [8.7,48.9,75.0]|
```

```
|[11.7,36.9,45.2]|  7.3|[11.7,36.9,45.2]|
+----------------+-----+----------------+
only showing top 5 rows

+---------------+-----+---------------+
|       features|label|indexedFeatures|
+---------------+-----+---------------+
| [0.7,39.6,8.7]|  1.6| [0.7,39.6,8.7]|
| [4.1,11.6,5.7]|  3.2| [4.1,11.6,5.7]|
|[7.3,28.1,41.4]|  5.5|[7.3,28.1,41.4]|
|  [8.6,2.1,1.0]|  4.8|  [8.6,2.1,1.0]|
|[17.2,4.1,31.6]|  5.9|[17.2,4.1,31.6]|
+---------------+-----+---------------+
only showing top 5 rows
```

7. Fit Generalized Linear Regression Model

```
# Import LinearRegression class
from pyspark.ml.regression import GeneralizedLinearRegression

# Define LinearRegression algorithm
glr = GeneralizedLinearRegression(family="gaussian", link="identity",\
                                  maxIter=10, regParam=0.3)
```

8. Pipeline Architecture

```
# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, glr])

model = pipeline.fit(trainingData)
```

9. Summary of the Model

Spark has a poor summary function for data and model. I wrote a summary function which has similar format as **R** output for the linear regression in PySpark.

```
def modelsummary(model):
    import numpy as np
    print ("Note: the last rows are the information for Intercept")
    print ("##","------------------------------------------------")
    print ("##","  Estimate   |   Std.Error | t Values  |  P-value")
    coef = np.append(list(model.coefficients),model.intercept)
    Summary=model.summary

    for i in range(len(Summary.pValues)):
        print ("##",'{:10.6f}'.format(coef[i]),\
        '{:10.6f}'.format(Summary.coefficientStandardErrors[i]),\
        '{:8.3f}'.format(Summary.tValues[i]),\
        '{:10.6f}'.format(Summary.pValues[i]))

    print ("##",'---')
#     print ("##","Mean squared error: % .6f" \
#             % Summary.meanSquaredError, ", RMSE: % .6f" \
#             % Summary.rootMeanSquaredError )
#     print ("##","Multiple R-squared: %f" % Summary.r2, ", \
#              Total iterations: %i"% Summary.totalIterations)
```

```
modelsummary(model.stages[-1])
```

You will get the following summary results:

```
Note: the last rows are the information for Intercept
('##', '-------------------------------------------------')
('##', '  Estimate   |   Std.Error | t Values  |  P-value')
('##', '  0.042857', '  0.001668', '  25.692', '  0.000000')
('##', '  0.199922', '  0.009881', '  20.232', '  0.000000')
('##', ' -0.001957', '  0.006917', '  -0.283', '  0.777757')
('##', '  3.007515', '  0.406389', '   7.401', '  0.000000')
('##', '---')
```

10. Make predictions

```
# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
+---------------+-----+------------------+
|       features|label|        prediction|
+---------------+-----+------------------+
| [0.7,39.6,8.7]|  1.6|10.937383732327625|
| [4.1,11.6,5.7]|  3.2| 5.491166258750164|
|[7.3,28.1,41.4]|  5.5|   8.8571603947873|
|  [8.6,2.1,1.0]|  4.8| 3.793966281660073|
|[17.2,4.1,31.6]|  5.9| 4.502507124763654|
+---------------+-----+------------------+
only showing top 5 rows
```

11. Evaluation

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.evaluation import RegressionEvaluator
# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(labelCol="label",
                                predictionCol="prediction",
                                metricName="rmse")

rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

The final Root Mean Squared Error (RMSE) is as follows:

```
Root Mean Squared Error (RMSE) on test data = 1.89857

y_true = predictions.select("label").toPandas()
y_pred = predictions.select("prediction").toPandas()

import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {0}'.format(r2_score))
```

Then you will get the R^2 value:

```
r2_score: 0.87707391843
```

8.3 Decision tree Regression

8.3.1 Introduction

8.3.2 How to solve it?

8.3.3 Demo

- The Jupyter notebook can be download from Decision Tree Regression.
- For more details about the parameters, please visit Decision Tree Regressor API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark regression example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                        options(header='true', \
                        inferschema='true').\
            load("../data/Advertising.csv",header=True);
```

check the data set

```
df.show(5,True)
df.printSchema()
```

Then you will get

```
+-----+-----+---------+-----+
|   TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
| 17.2| 45.9|     69.3|  9.3|
|151.5| 41.3|     58.5| 18.5|
|180.8| 10.8|     58.4| 12.9|
+-----+-----+---------+-----+
only showing top 5 rows

root
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

You can also get the Statistical resutls from the data frame (Unfortunately, it only works for numerical).

```
df.describe().show()
```

Then you will get

```
+-------+-----------------+------------------+------------------+------------------+
|summary|               TV|             Radio|         Newspaper|             Sales|
+-------+-----------------+------------------+------------------+------------------+
|  count|              200|               200|               200|               200|
|   mean|         147.0425|23.264000000000024|30.553999999999995|14.022500000000003|
| stddev|85.85423631490805|14.846809176168728| 21.77862083852283| 5.217456565710477|
|    min|              0.7|               0.0|               0.3|               1.6|
|    max|            296.4|              49.6|             114.0|              27.0|
+-------+-----------------+------------------+------------------+------------------+
```

3. Convert the data to dense vector (**features** and **label**)

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for unsupervised learn
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix
```

```
        :author: Wenqiang Feng
        :email:  von198@gmail.com
        '''

        indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                     for c in categoricalCols ]

        # default setting: dropLast=True
        encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                     outputCol="{0}_encoded".format(indexer.getOutputCol()))
                     for indexer in indexers ]

        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
```

```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# I provide two ways to build the features and labels

# method 1 (good for small feature):
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
#                                       row["Radio"],
#                                       row["Newspaper"]]))

# Method 2 (good for large features):
def transData(data):
return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label'])

transformed= transData(df)
transformed.show(5)
```

```
+-----------------+-----+
|         features|label|
+-----------------+-----+
|[230.1,37.8,69.2]| 22.1|
| [44.5,39.3,45.1]| 10.4|
| [17.2,45.9,69.3]|  9.3|
|[151.5,41.3,58.5]| 18.5|
|[180.8,10.8,58.4]| 12.9|
+-----------------+-----+
only showing top 5 rows
```

Note: You will find out that all of the machine learning algorithms in Spark are based on the **features** and **label**. That is to say, you can play with all of the machine learning algorithms in Spark when you get ready the **features** and **label**.

4. Convert the data to dense vector

```
# convert the data to dense vector
def transData(data):
    return data.rdd.map(lambda r: [r[-1], Vectors.dense(r[:-1])]).\
           toDF(['label','features'])

transformed = transData(df)
transformed.show(5)
```

5. Deal with the Categorical variables

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4
# distinct values are treated as continuous.

featureIndexer = VectorIndexer(inputCol="features", \
                               outputCol="indexedFeatures",\
                               maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
```

When you check you data at this point, you will get

```
+-----------------+-----+-----------------+
|         features|label|  indexedFeatures|
+-----------------+-----+-----------------+
|[230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2]|
| [44.5,39.3,45.1]| 10.4| [44.5,39.3,45.1]|
| [17.2,45.9,69.3]|  9.3| [17.2,45.9,69.3]|
|[151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5]|
|[180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4]|
+-----------------+-----+-----------------+
only showing top 5 rows
```

6. Split the data into training and test sets (40% held out for testing)

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])
```

You can check your train and test data as follows (In my opinion, it is always to good to keep tracking your data during prototype pahse):

```
trainingData.show(5)
testData.show(5)
```

Then you will get

```
+---------------+-----+---------------+
|       features|label|indexedFeatures|
+---------------+-----+---------------+
| [4.1,11.6,5.7]|  3.2| [4.1,11.6,5.7]|
|[7.3,28.1,41.4]|  5.5|[7.3,28.1,41.4]|
| [8.4,27.2,2.1]|  5.7| [8.4,27.2,2.1]|
```

```
|  [8.6,2.1,1.0]|  4.8|  [8.6,2.1,1.0]|
|[8.7,48.9,75.0]|  7.2|[8.7,48.9,75.0]|
+---------------+-----+---------------+
only showing top 5 rows

+---------------+-----+---------------+
|       features|label| indexedFeatures|
+---------------+-----+---------------+
|  [0.7,39.6,8.7]|  1.6|  [0.7,39.6,8.7]|
|  [5.4,29.9,9.4]|  5.3|  [5.4,29.9,9.4]|
| [7.8,38.9,50.6]|  6.6| [7.8,38.9,50.6]|
|[17.2,45.9,69.3]|  9.3|[17.2,45.9,69.3]|
|[18.7,12.1,23.4]|  6.7|[18.7,12.1,23.4]|
+---------------+-----+---------------+
only showing top 5 rows
```

7. Fit Decision Tree Regression Model

```
from pyspark.ml.regression import DecisionTreeRegressor

# Train a DecisionTree model.
dt = DecisionTreeRegressor(featuresCol="indexedFeatures")
```

8. Pipeline Architecture

```
# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, dt])

model = pipeline.fit(trainingData)
```

9. Make predictions

```
# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
+----------+-----+---------------+
|prediction|label|       features|
+----------+-----+---------------+
|       7.2|  1.6|  [0.7,39.6,8.7]|
|       7.3|  5.3|  [5.4,29.9,9.4]|
|       7.2|  6.6| [7.8,38.9,50.6]|
|      8.64|  9.3|[17.2,45.9,69.3]|
|      6.45|  6.7|[18.7,12.1,23.4]|
+----------+-----+---------------+
only showing top 5 rows
```

10. Evaluation

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.evaluation import RegressionEvaluator
# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(labelCol="label",
                                predictionCol="prediction",
                                metricName="rmse")
```

```
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

The final Root Mean Squared Error (RMSE) is as follows:

```
Root Mean Squared Error (RMSE) on test data = 1.50999

y_true = predictions.select("label").toPandas()
y_pred = predictions.select("prediction").toPandas()

import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {0}'.format(r2_score))
```

Then you will get the R^2 value:

```
r2_score: 0.911024318967
```

You may also check the importance of the features:

```
model.stages[1].featureImportances
```

The you will get the weight for each features

```
SparseVector(3, {0: 0.6811, 1: 0.3187, 2: 0.0002})
```

8.4 Random Forest Regression

8.4.1 Introduction

8.4.2 How to solve it?

8.4.3 Demo

- The Jupyter notebook can be download from Random Forest Regression.
- For more details about the parameters, please visit Random Forest Regressor API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark RandomForest Regression example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                               options(header='true', \
                               inferschema='true').\
                               load("../data/Advertising.csv",header=True);
```

```
df.show(5,True)
df.printSchema()
```

```
+-----+-----+---------+-----+
|   TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
| 17.2| 45.9|     69.3|  9.3|
|151.5| 41.3|     58.5| 18.5|
|180.8| 10.8|     58.4| 12.9|
+-----+-----+---------+-----+
only showing top 5 rows

root
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)

df.describe().show()

+-------+-----------------+------------------+------------------+------------------+
|summary|               TV|             Radio|         Newspaper|             Sales|
+-------+-----------------+------------------+------------------+------------------+
|  count|              200|               200|               200|               200|
|   mean|         147.0425|23.264000000000024|30.553999999999995|14.022500000000003|
| stddev|85.85423631490805|14.846809176168728| 21.77862083852283| 5.217456565710477|
|    min|              0.7|               0.0|               0.3|               1.6|
|    max|            296.4|              49.6|             114.0|              27.0|
+-------+-----------------+------------------+------------------+------------------+
```

3. Convert the data to dense vector (**features** and **label**)

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])
```

```
        model=pipeline.fit(df)
        data = model.transform(df)

        data = data.withColumn('label',col(labelCol))

        return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
    def get_dummy(df,indexCol,categoricalCols,continuousCols):
        '''
        Get dummy variables and concat with continuous variables for unsupervised learn
        :param df: the dataframe
        :param categoricalCols: the name list of the categorical data
        :param continuousCols:  the name list of the numerical data
        :return k: feature matrix

        :author: Wenqiang Feng
        :email:  von198@gmail.com
        '''

        indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                     for c in categoricalCols ]

        # default setting: dropLast=True
        encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                     outputCol="{0}_encoded".format(indexer.getOutputCol()))
                     for indexer in indexers ]

        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
```

```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# convert the data to dense vector
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
#                                       row["Radio"],
#                                       row["Newspaper"]]))
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label
```

4. Convert the data to dense vector

```
transformed= transData(df)
transformed.show(5)
```

```
+-----------------+-----+
|         features|label|
+-----------------+-----+
|[230.1,37.8,69.2]| 22.1|
| [44.5,39.3,45.1]| 10.4|
| [17.2,45.9,69.3]|  9.3|
|[151.5,41.3,58.5]| 18.5|
|[180.8,10.8,58.4]| 12.9|
+-----------------+-----+
only showing top 5 rows
```

5. Deal with the Categorical variables

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

featureIndexer = VectorIndexer(inputCol="features", \
                               outputCol="indexedFeatures",\
                               maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
data.show(5,True)
```

```
+-----------------+-----+-----------------+
|         features|label|  indexedFeatures|
+-----------------+-----+-----------------+
|[230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2]|
| [44.5,39.3,45.1]| 10.4| [44.5,39.3,45.1]|
| [17.2,45.9,69.3]|  9.3| [17.2,45.9,69.3]|
|[151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5]|
|[180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4]|
+-----------------+-----+-----------------+
only showing top 5 rows
```

6. Split the data into training and test sets (40% held out for testing)

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5)
testData.show(5)
```

```
+----------------+-----+----------------+
|        features|label| indexedFeatures|
+----------------+-----+----------------+
|  [0.7,39.6,8.7]|  1.6|  [0.7,39.6,8.7]|
|   [8.6,2.1,1.0]|  4.8|   [8.6,2.1,1.0]|
| [8.7,48.9,75.0]|  7.2| [8.7,48.9,75.0]|
|[11.7,36.9,45.2]|  7.3|[11.7,36.9,45.2]|
|[13.2,15.9,49.6]|  5.6|[13.2,15.9,49.6]|
+----------------+-----+----------------+
only showing top 5 rows

+---------------+-----+---------------+
|       features|label|indexedFeatures|
+---------------+-----+---------------+
```

```
| [4.1,11.6,5.7]|  3.2| [4.1,11.6,5.7]|
| [5.4,29.9,9.4]|  5.3| [5.4,29.9,9.4]|
|[7.3,28.1,41.4]|  5.5|[7.3,28.1,41.4]|
|[7.8,38.9,50.6]|  6.6|[7.8,38.9,50.6]|
| [8.4,27.2,2.1]|  5.7| [8.4,27.2,2.1]|
+---------------+-----+---------------+
only showing top 5 rows
```

7. Fit RandomForest Regression Model

```
# Import LinearRegression class
from pyspark.ml.regression import RandomForestRegressor

# Define LinearRegression algorithm
rf = RandomForestRegressor() # featuresCol="indexedFeatures",numTrees=2, maxDepth=2, see
```

Note: If you decide to use the `indexedFeatures` features, you need to add the parameter `featuresCol="indexedFeatures"`.

---

8. Pipeline Architecture

```
# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, rf])
model = pipeline.fit(trainingData)
```

9. Make predictions

```
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("features","label", "prediction").show(5)
```

```
+---------------+-----+------------------+
|       features|label|        prediction|
+---------------+-----+------------------+
| [4.1,11.6,5.7]|  3.2| 8.155439814814816|
| [5.4,29.9,9.4]|  5.3|10.412769901394899|
|[7.3,28.1,41.4]|  5.5| 12.13735648148148|
|[7.8,38.9,50.6]|  6.6|11.321796703296704|
| [8.4,27.2,2.1]|  5.7|12.071421957671957|
+---------------+-----+------------------+
only showing top 5 rows
```

10. Evaluation

```
# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

Root Mean Squared Error (RMSE) on test data = 2.35912

import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {:4.3f}'.format(r2_score))
```

```
r2_score: 0.831
```

11. Feature importances

```
model.stages[-1].featureImportances
```

```
SparseVector(3, {0: 0.4994, 1: 0.3196, 2: 0.181})
```

```
model.stages[-1].trees
```

```
[DecisionTreeRegressionModel (uid=dtr_c75f1c75442c) of depth 5 with 43 nodes,
 DecisionTreeRegressionModel (uid=dtr_70fc2d441581) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_bc8464f545a7) of depth 5 with 31 nodes,
 DecisionTreeRegressionModel (uid=dtr_a8a7e5367154) of depth 5 with 59 nodes,
 DecisionTreeRegressionModel (uid=dtr_3ea01314fcbc) of depth 5 with 47 nodes,
 DecisionTreeRegressionModel (uid=dtr_be9a04ac22a6) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_38610d47328a) of depth 5 with 51 nodes,
 DecisionTreeRegressionModel (uid=dtr_bf14aea0ad3b) of depth 5 with 49 nodes,
 DecisionTreeRegressionModel (uid=dtr_cde24ebd6bb6) of depth 5 with 39 nodes,
 DecisionTreeRegressionModel (uid=dtr_a1fc9bd4fbeb) of depth 5 with 57 nodes,
 DecisionTreeRegressionModel (uid=dtr_37798d6db1ba) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_c078b73ada63) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_fd00e3a070ad) of depth 5 with 55 nodes,
 DecisionTreeRegressionModel (uid=dtr_9d01d5fb8604) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_8bd8bdddf642) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_e53b7bae30f8) of depth 5 with 49 nodes,
 DecisionTreeRegressionModel (uid=dtr_808a869db21c) of depth 5 with 47 nodes,
 DecisionTreeRegressionModel (uid=dtr_64d0916bceb0) of depth 5 with 33 nodes,
 DecisionTreeRegressionModel (uid=dtr_0891055fff94) of depth 5 with 55 nodes,
 DecisionTreeRegressionModel (uid=dtr_19c8bbad26c2) of depth 5 with 51 nodes]
```

8.5 Gradient-boosted tree regression

8.5.1 Introduction

8.5.2 How to solve it?

8.5.3 Demo

- The Jupyter notebook can be download from Gradient-boosted tree regression.
- For more details about the parameters, please visit Gradient boosted tree API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession \
    .builder \
    .appName("Python Spark GBTRegressor example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                                options(header='true', \
                                inferschema='true').\
                                load("../data/Advertising.csv",header=True);

df.show(5,True)
df.printSchema()
```

```
+-----+-----+---------+-----+
|   TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
| 17.2| 45.9|     69.3|  9.3|
|151.5| 41.3|     58.5| 18.5|
|180.8| 10.8|     58.4| 12.9|
+-----+-----+---------+-----+
only showing top 5 rows

root
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)

df.describe().show()
```

```
+-------+-----------------+------------------+------------------+------------------+
|summary|               TV|             Radio|         Newspaper|             Sales|
+-------+-----------------+------------------+------------------+------------------+
|  count|              200|               200|               200|               200|
|   mean|         147.0425|23.264000000000024|30.553999999999995|14.022500000000003|
| stddev|85.85423631490805|14.846809176168728| 21.77862083852283| 5.217456565710477|
|    min|              0.7|               0.0|               0.3|               1.6|
|    max|            296.4|              49.6|             114.0|              27.0|
+-------+-----------------+------------------+------------------+------------------+
```

3. Convert the data to dense vector (**features** and **label**)

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]
```

```
        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        data = data.withColumn('label',col(labelCol))

        return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
    def get_dummy(df,indexCol,categoricalCols,continuousCols):
        '''
        Get dummy variables and concat with continuous variables for unsupervised learn
        :param df: the dataframe
        :param categoricalCols: the name list of the categorical data
        :param continuousCols:  the name list of the numerical data
        :return k: feature matrix

        :author: Wenqiang Feng
        :email:  von198@gmail.com
        '''

        indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                     for c in categoricalCols ]

        # default setting: dropLast=True
        encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                     outputCol="{0}_encoded".format(indexer.getOutputCol()))
                     for indexer in indexers ]

        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
```

```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# convert the data to dense vector
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
#                                       row["Radio"],
#                                       row["Newspaper"]]))
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label
```

4. Convert the data to dense vector

```
transformed= transData(df)
transformed.show(5)
```

```
+-----------------+-----+
|         features|label|
+-----------------+-----+
|[230.1,37.8,69.2]| 22.1|
| [44.5,39.3,45.1]| 10.4|
| [17.2,45.9,69.3]|  9.3|
|[151.5,41.3,58.5]| 18.5|
|[180.8,10.8,58.4]| 12.9|
+-----------------+-----+
only showing top 5 rows
```

5. Deal with the Categorical variables

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

featureIndexer = VectorIndexer(inputCol="features", \
                               outputCol="indexedFeatures",\
                               maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
data.show(5,True)
```

```
+-----------------+-----+-----------------+
|         features|label|  indexedFeatures|
+-----------------+-----+-----------------+
|[230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2]|
| [44.5,39.3,45.1]| 10.4| [44.5,39.3,45.1]|
| [17.2,45.9,69.3]|  9.3| [17.2,45.9,69.3]|
|[151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5]|
|[180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4]|
+-----------------+-----+-----------------+
only showing top 5 rows
```

6. Split the data into training and test sets (40% held out for testing)

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5)
testData.show(5)
```

```
+----------------+-----+----------------+
|        features|label| indexedFeatures|
+----------------+-----+----------------+
|  [0.7,39.6,8.7]|  1.6|  [0.7,39.6,8.7]|
|   [8.6,2.1,1.0]|  4.8|   [8.6,2.1,1.0]|
| [8.7,48.9,75.0]|  7.2| [8.7,48.9,75.0]|
|[11.7,36.9,45.2]|  7.3|[11.7,36.9,45.2]|
|[13.2,15.9,49.6]|  5.6|[13.2,15.9,49.6]|
+----------------+-----+----------------+
only showing top 5 rows
```

```
+---------------+-----+---------------+
|       features|label|indexedFeatures|
+---------------+-----+---------------+
| [4.1,11.6,5.7]|  3.2| [4.1,11.6,5.7]|
| [5.4,29.9,9.4]|  5.3| [5.4,29.9,9.4]|
|[7.3,28.1,41.4]|  5.5|[7.3,28.1,41.4]|
|[7.8,38.9,50.6]|  6.6|[7.8,38.9,50.6]|
| [8.4,27.2,2.1]|  5.7| [8.4,27.2,2.1]|
+---------------+-----+---------------+
only showing top 5 rows
```

7. Fit RandomForest Regression Model

```
# Import LinearRegression class
from pyspark.ml.regression import GBTRegressor

# Define LinearRegression algorithm
rf = GBTRegressor() #numTrees=2, maxDepth=2, seed=42
```

Note: If you decide to use the `indexedFeatures` features, you need to add the parameter `featuresCol="indexedFeatures"`.

---

8. Pipeline Architecture

```
# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, rf])
model = pipeline.fit(trainingData)
```

9. Make predictions

```
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("features","label", "prediction").show(5)
```

```
+---------------+-----+------------------+
|       features|label|        prediction|
+---------------+-----+------------------+
| [7.8,38.9,50.6]|  6.6| 6.836040343319862|
|   [8.6,2.1,1.0]|  4.8| 5.652202764688849|
| [8.7,48.9,75.0]|  7.2| 6.908750296855572|
| [13.1,0.4,25.6]|  5.3| 5.784020210692574|
|[19.6,20.1,17.0]|  7.6|6.8678921062629295|
+---------------+-----+------------------+
only showing top 5 rows
```

10. Evaluation

```
# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
Root Mean Squared Error (RMSE) on test data = 1.36939
```

```
import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {:4.3f}'.format(r2_score))

r2_score: 0.932
```

11. Feature importances

```
model.stages[-1].featureImportances

SparseVector(3, {0: 0.3716, 1: 0.3525, 2: 0.2759})

model.stages[-1].trees

[DecisionTreeRegressionModel (uid=dtr_7f5cd2ef7cb6) of depth 5 with 61 nodes,
 DecisionTreeRegressionModel (uid=dtr_ef3ab6baeac9) of depth 5 with 39 nodes,
 DecisionTreeRegressionModel (uid=dtr_07c6e3cf3819) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_ce724af79a2b) of depth 5 with 47 nodes,
 DecisionTreeRegressionModel (uid=dtr_d149ecc71658) of depth 5 with 55 nodes,
 DecisionTreeRegressionModel (uid=dtr_d3a79bdea516) of depth 5 with 43 nodes,
 DecisionTreeRegressionModel (uid=dtr_7abc1a337844) of depth 5 with 51 nodes,
 DecisionTreeRegressionModel (uid=dtr_480834b46d8f) of depth 5 with 33 nodes,
 DecisionTreeRegressionModel (uid=dtr_0cbd1eaa3874) of depth 5 with 39 nodes,
 DecisionTreeRegressionModel (uid=dtr_8088ac71a204) of depth 5 with 57 nodes,
 DecisionTreeRegressionModel (uid=dtr_2ceb9e8deb45) of depth 5 with 47 nodes,
 DecisionTreeRegressionModel (uid=dtr_cc334e84e9a2) of depth 5 with 57 nodes,
 DecisionTreeRegressionModel (uid=dtr_a665c562929e) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_2999b1ffd2dc) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_29965cbe8cfc) of depth 5 with 55 nodes,
 DecisionTreeRegressionModel (uid=dtr_731df51bf0ad) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_354cf33424da) of depth 5 with 51 nodes,
 DecisionTreeRegressionModel (uid=dtr_4230f200b1c0) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_3279cdc1ce1d) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_f474a99ff06e) of depth 5 with 55 nodes]
```

CHAPTER

NINE

REGULARIZATION

In mathematics, statistics, and computer science, particularly in the fields of machine learning and inverse problems, regularization is a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting (Wikipedia Regularization).

Due to the sparsity within our data, our training sets will often be ill-posed (singular). Applying regularization to the regression has many advantages, including:

1. Converting ill-posed problems to well-posed by adding additional information via the penalty parameter λ
2. Preventing overfitting
3. Variable selection and the removal of correlated variables (Glmnet Vignette). The Ridge method shrinks the coefficients of correlated variables while the LASSO method picks one variable and discards the others. The elastic net penalty is a mixture of these two; if variables are correlated in groups then $\alpha = 0.5$ tends to select the groups as in or out. If α is close to 1, the elastic net performs much like the LASSO method and removes any degeneracies and wild behavior caused by extreme correlations.

9.1 Ridge regression

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \|\hat{X}\beta - \hat{Y}\|^2 + \lambda \|\beta\|_2^2$$

9.2 Least Absolute Shrinkage and Selection Operator (LASSO)

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \|\hat{X}\beta - \hat{Y}\|^2 + \lambda \|\beta\|_1$$

9.3 Elastic net

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \|\hat{X}\beta - \hat{Y}\|^2 + \lambda(\alpha\|\beta\|_1 + (1-\alpha)\|\beta\|_2^2), \alpha \in [0,1]$$

CHAPTER

TEN

CLASSIFICATION

Note: Birds of a feather folock together. – old Chinese proverb

10.1 Binomial logistic regression

10.1.1 Introduction

10.1.2 Demo

- The Jupyter notebook can be download from Logistic Regression.
- For more details, please visit Logistic Regression API .

Note: In this demo, I introduced a new function `get_dummy` to deal with the categorical data. I highly recommend you to use my `get_dummy` function in the other cases. This function will save a lot of time for you.

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

        spark = SparkSession \
            .builder \
            .appName("Python Spark Logistic Regression example") \
            .config("spark.some.config.option", "some-value") \
            .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv') \
            .options(header='true', inferschema='true') \
            .load("./data/bank.csv",header=True);
df.drop('day','month','poutcome').show(5)
```

```
+---+-----------+-------+---------+-------+-------+-------+----+-------+--------+------
|age|        job|marital|education|default|balance|housing|loan|contact|duration|campai
+---+-----------+-------+---------+-------+-------+-------+----+-------+--------+------
| 58|  management|married| tertiary|     no|   2143|    yes|  no|unknown|     261|
| 44|  technician| single|secondary|     no|     29|    yes|  no|unknown|     151|
| 33|entrepreneur|married|secondary|     no|      2|    yes| yes|unknown|      76|
| 47| blue-collar|married|  unknown|     no|   1506|    yes|  no|unknown|      92|
```

```
| 33|     unknown| single|  unknown|     no|      1|     no|  no|unknown|     198|
+---+-----------+-------+---------+-------+-------+-------+----+-------+--------+------
only showing top 5 rows

df.printSchema()

root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- y: string (nullable = true)
```

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
    def get_dummy(df,indexCol,categoricalCols,continuousCols):
        '''
        Get dummy variables and concat with continuous variables for unsupervised learn
        :param df: the dataframe
        :param categoricalCols: the name list of the categorical data
        :param continuousCols:  the name list of the numerical data
        :return k: feature matrix

        :author: Wenqiang Feng
        :email:  von198@gmail.com
        '''

        indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                     for c in categoricalCols ]

        # default setting: dropLast=True
        encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                     outputCol="{0}_encoded".format(indexer.getOutputCol()))
                     for indexer in indexers ]

        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
```

```
def get_dummy(df,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in encoder
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select('features','label')
```

3. Deal with categorical data and Convert the data to dense vector

```
catcols = ['job','marital','education','default',
           'housing','loan','contact','poutcome']

num_cols = ['balance', 'duration','campaign','pdays','previous',]
labelCol = 'y'

data = get_dummy(df,catcols,num_cols,labelCol)
data.show(5)
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|(29,[1,11,14,16,1...|   no|
|(29,[2,12,13,16,1...|   no|
|(29,[7,11,13,16,1...|   no|
|(29,[0,11,16,17,1...|   no|
|(29,[12,16,18,20,...|   no|
+--------------------+-----+
only showing top 5 rows
```

4. Deal with Categorical Label and Variables

```
from pyspark.ml.feature import StringIndexer
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(data)
labelIndexer.transform(data).show(5, True)
```

```
+--------------------+-----+------------+
|            features|label|indexedLabel|
+--------------------+-----+------------+
|(29,[1,11,14,16,1...|   no|         0.0|
|(29,[2,12,13,16,1...|   no|         0.0|
|(29,[7,11,13,16,1...|   no|         0.0|
|(29,[0,11,16,17,1...|   no|         0.0|
|(29,[12,16,18,20,...|   no|         0.0|
+--------------------+-----+------------+
only showing top 5 rows
```

```
from pyspark.ml.feature import VectorIndexer
# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated as continuous.
featureIndexer =VectorIndexer(inputCol="features", \
                              outputCol="indexedFeatures", \
                              maxCategories=4).fit(data)
featureIndexer.transform(data).show(5, True)
```

```
+--------------------+-----+--------------------+
|            features|label|     indexedFeatures|
+--------------------+-----+--------------------+
|(29,[1,11,14,16,1...|   no|(29,[1,11,14,16,1...|
|(29,[2,12,13,16,1...|   no|(29,[2,12,13,16,1...|
|(29,[7,11,13,16,1...|   no|(29,[7,11,13,16,1...|
|(29,[0,11,16,17,1...|   no|(29,[0,11,16,17,1...|
|(29,[12,16,18,20,...|   no|(29,[12,16,18,20,...|
+--------------------+-----+--------------------+
only showing top 5 rows
```

5. Split the data to training and test data sets

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5,False)
testData.show(5,False)
```

```
+-----------------------------------------------------------------------------------------------------------
|features
+-----------------------------------------------------------------------------------------------------------
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-731.0,401.0,
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-723.0,112.0,
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-626.0,205.0,
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-498.0,357.0,
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-477.0,473.0,
+-----------------------------------------------------------------------------------------------------------
only showing top 5 rows

+-----------------------------------------------------------------------------------------------------------
|features
+-----------------------------------------------------------------------------------------------------------
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-648.0,280.0,
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-596.0,147.0,
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-529.0,416.0,
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-518.0,46.0,5
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-470.0,275.0,
+-----------------------------------------------------------------------------------------------------------
only showing top 5 rows
```

6. Fit Logistic Regression Model

```
from pyspark.ml.classification import LogisticRegression
logr = LogisticRegression(featuresCol='indexedFeatures', labelCol='indexedLabel')
```

7. Pipeline Architecture

```
# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, logr,labelConverter])

# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

8. Make predictions

```
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

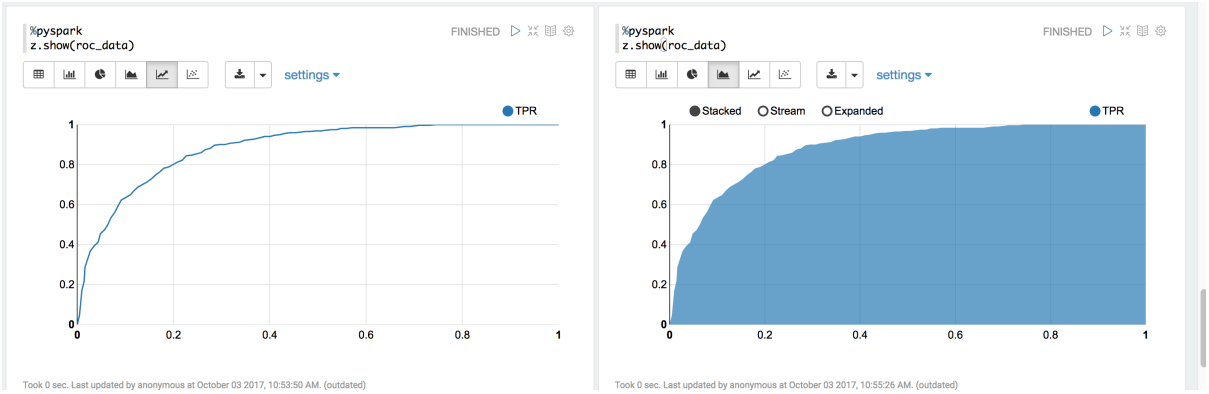
```
+--------------------+-----+--------------+
|            features|label|predictedLabel|
+--------------------+-----+--------------+
|(29,[0,11,13,16,1...|   no|            no|
|(29,[0,11,13,16,1...|   no|            no|
```

```
|(29,[0,11,13,16,1...|   no|            no|
|(29,[0,11,13,16,1...|   no|            no|
|(29,[0,11,13,16,1...|   no|            no|
+--------------------+-----+--------------+
only showing top 5 rows
```

9. Evaluation

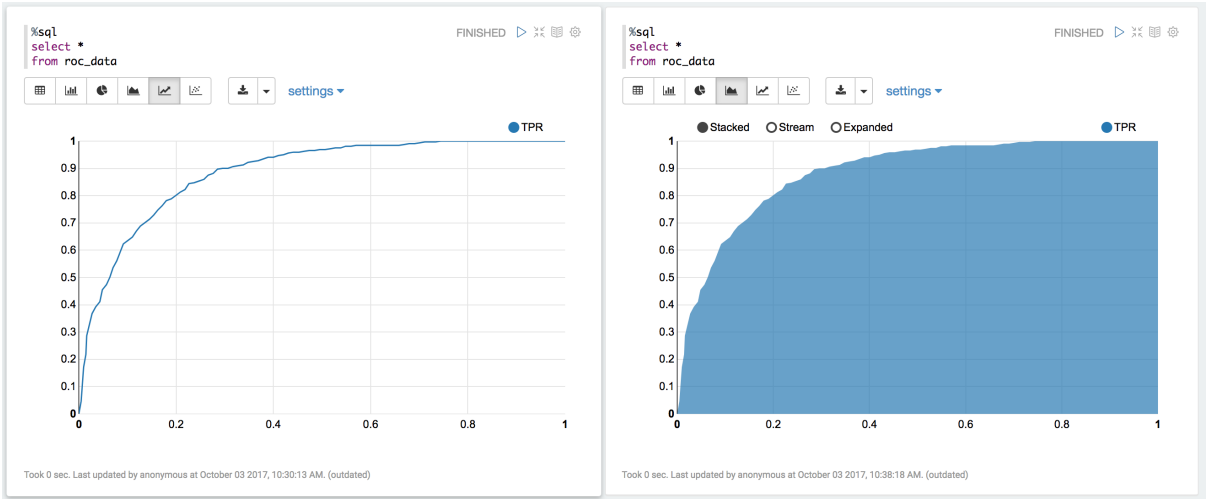
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))

Test Error = 0.0987688

lrModel = model.stages[2]
trainingSummary = lrModel.summary

# Obtain the objective per iteration
# objectiveHistory = trainingSummary.objectiveHistory
# print("objectiveHistory:")
# for objective in objectiveHistory:
#     print(objective)

# Obtain the receiver-operating characteristic as a dataframe and areaUnderROC.
trainingSummary.roc.show(5)
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))

# Set the model threshold to maximize F-Measure
fMeasure = trainingSummary.fMeasureByThreshold
maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').head(5)
# bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-Measure)'])
#     .select('threshold').head()['threshold']
# lr.setThreshold(bestThreshold)
```

You can use `z.show()` to get the data and plot the ROC curves:



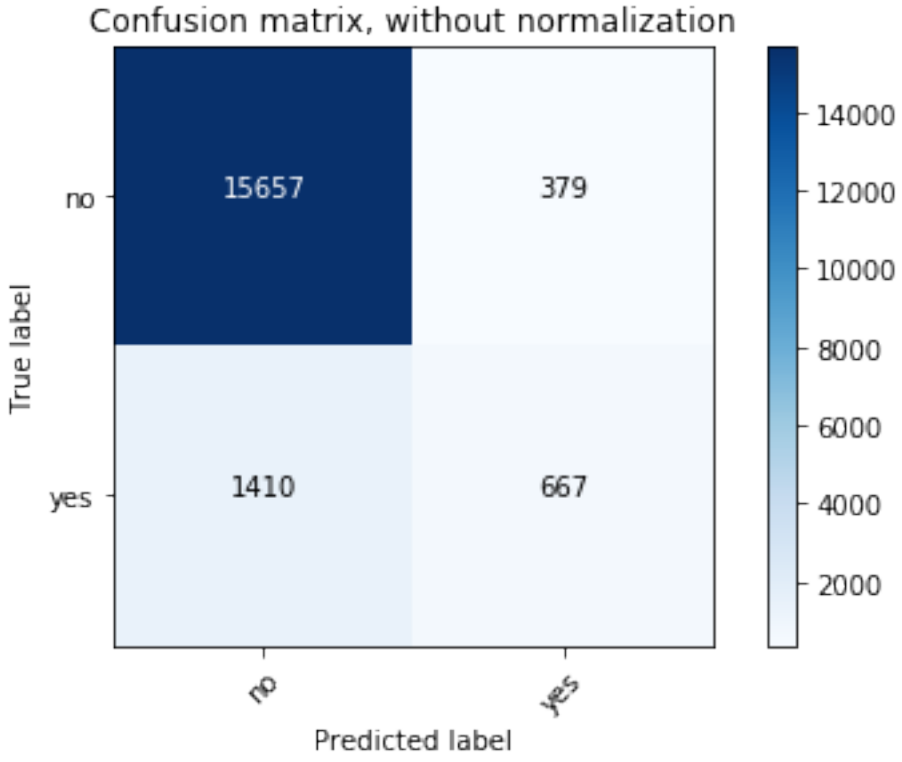
You can also register a TempTable `data.registerTempTable('roc_data')` and then use `sql` to plot the ROC curve:

10. visualization



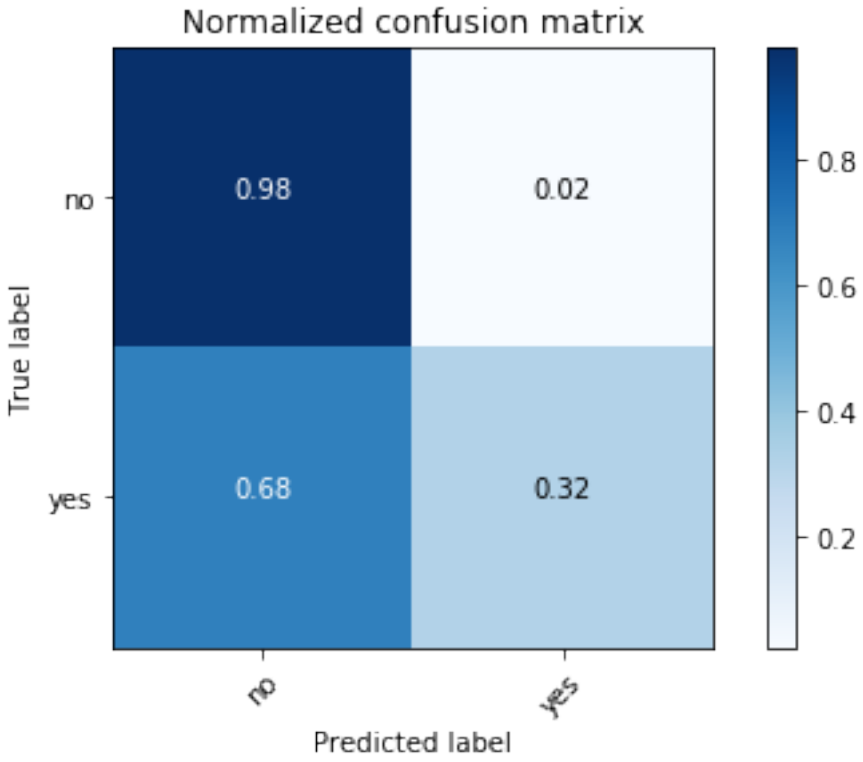

```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

class_temp = predictions.select("label").groupBy("label")\
                        .count().sort('count', ascending=False).toPandas()
```

```
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names

['no', 'yes']

from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix

array([[15657,   379],
       [ 1410,   667]])

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()

Confusion matrix, without normalization
[[15657   379]
 [ 1410   667]]
```



```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[ 0.97636568  0.02363432]
 [ 0.67886375  0.32113625]]
```



10.2 Multinomial logistic regression

10.2.1 Introduction

10.2.2 Demo

- The Jupyter notebook can be download from Logistic Regression.
- For more details, please visit Logistic Regression API .

Note: In this demo, I introduced a new function `get_dummy` to deal with the categorical data. I highly recommend you to use my `get_dummy` function in the other cases. This function will save a lot of time for you.

---

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark MultinomialLogisticRegression classification") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv') \
            .options(header='true', inferschema='true') \
            .load("./data/WineData2.csv",header=True);
df.show(5)
```

```
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|fixed|volatile|citric|sugar|chlorides|free|total|density|  pH|sulphates|alcohol|quality
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4|      5
|  7.8|    0.88|   0.0|  2.6|    0.098|25.0| 67.0| 0.9968| 3.2|     0.68|    9.8|      5
|  7.8|    0.76|  0.04|  2.3|    0.092|15.0| 54.0|  0.997|3.26|     0.65|    9.8|      5
| 11.2|    0.28|  0.56|  1.9|    0.075|17.0| 60.0|  0.998|3.16|     0.58|    9.8|      6
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4|      5
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
only showing top 5 rows

df.printSchema()

root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)

# Convert to float format
def string_to_float(x):
    return float(x)

#
def condition(r):
    if (0<= r <= 4):
        label = "low"
    elif(4< r <= 6):
        label = "medium"
    else:
        label = "high"
    return label

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
```

```
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())

df = df.withColumn("quality", quality_udf("quality"))

df.show(5,True)
```

```
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|fixed|volatile|citric|sugar|chlorides|free|total|density|  pH|sulphates|alcohol|quality
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4| medium
|  7.8|    0.88|   0.0|  2.6|    0.098|25.0| 67.0| 0.9968| 3.2|     0.68|    9.8| medium
|  7.8|    0.76|  0.04|  2.3|    0.092|15.0| 54.0|  0.997|3.26|     0.65|    9.8| medium
| 11.2|    0.28|  0.56|  1.9|    0.075|17.0| 60.0|  0.998|3.16|     0.58|    9.8| medium
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4| medium
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
only showing top 5 rows
```

```
df.printSchema()
```

```
root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)
```

3. Deal with categorical data and Convert the data to dense vector

---

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                + continuousCols, outputCol="features")
```

```
        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        data = data.withColumn('label',col(labelCol))

        return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
    def get_dummy(df,indexCol,categoricalCols,continuousCols):
        '''
        Get dummy variables and concat with continuous variables for unsupervised learn
        :param df: the dataframe
        :param categoricalCols: the name list of the categorical data
        :param continuousCols:  the name list of the numerical data
        :return k: feature matrix

        :author: Wenqiang Feng
        :email:  von198@gmail.com
        '''

        indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                     for c in categoricalCols ]

        # default setting: dropLast=True
        encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                     outputCol="{0}_encoded".format(indexer.getOutputCol()))
                     for indexer in indexers ]

        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
```

```
def get_dummy(df,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in encoder
                                + continuousCols, outputCol="features")
```

```
    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select('features','label')
```

4. Transform the dataset to DataFrame

```
from pyspark.ml.linalg import Vectors # !!!!caution: not from pyspark.mllib.linalg impor
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString,StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

def transData(data):
return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label'])

transformed = transData(df)
transformed.show(5)
```

```
+--------------------+------+
|            features| label|
+--------------------+------+
|[7.4,0.7,0.0,1.9,...|medium|
|[7.8,0.88,0.0,2.6...|medium|
|[7.8,0.76,0.04,2....|medium|
|[11.2,0.28,0.56,1...|medium|
|[7.4,0.7,0.0,1.9,...|medium|
+--------------------+------+
only showing top 5 rows
```

4. Deal with Categorical Label and Variables

```
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+------------+
|            features| label|indexedLabel|
+--------------------+------+------------+
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
|[7.8,0.88,0.0,2.6...|medium|         0.0|
|[7.8,0.76,0.04,2....|medium|         0.0|
|[11.2,0.28,0.56,1...|medium|         0.0|
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
+--------------------+------+------------+
only showing top 5 rows
```

```
# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated as continuous.
featureIndexer =VectorIndexer(inputCol="features", \
                              outputCol="indexedFeatures", \
                              maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+--------------------+
|            features| label|     indexedFeatures|
+--------------------+------+--------------------+
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
|[7.8,0.88,0.0,2.6...|medium|[7.8,0.88,0.0,2.6...|
|[7.8,0.76,0.04,2....|medium|[7.8,0.76,0.04,2....|
|[11.2,0.28,0.56,1...|medium|[11.2,0.28,0.56,1...|
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
+--------------------+------+--------------------+
only showing top 5 rows
```

5. Split the data to training and test data sets

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5,False)
testData.show(5,False)
```

```
+-----------------------------------------------------------+------+
|features                                                   |label |
+-----------------------------------------------------------+------+
|[4.7,0.6,0.17,2.3,0.058,17.0,106.0,0.9932,3.85,0.6,12.9]   |medium|
|[5.0,0.38,0.01,1.6,0.048,26.0,60.0,0.99084,3.7,0.75,14.0]|medium|
|[5.0,0.4,0.5,4.3,0.046,29.0,80.0,0.9902,3.49,0.66,13.6]   |medium|
|[5.0,0.74,0.0,1.2,0.041,16.0,46.0,0.99258,4.01,0.59,12.5]|medium|
|[5.1,0.42,0.0,1.8,0.044,18.0,88.0,0.99157,3.68,0.73,13.6]|high  |
+-----------------------------------------------------------+------+
only showing top 5 rows
```

```
+-----------------------------------------------------------+------+
|features                                                   |label |
+-----------------------------------------------------------+------+
|[4.6,0.52,0.15,2.1,0.054,8.0,65.0,0.9934,3.9,0.56,13.1]    |low   |
|[4.9,0.42,0.0,2.1,0.048,16.0,42.0,0.99154,3.71,0.74,14.0]|high  |
|[5.0,0.42,0.24,2.0,0.06,19.0,50.0,0.9917,3.72,0.74,14.0]  |high  |
|[5.0,1.02,0.04,1.4,0.045,41.0,85.0,0.9938,3.75,0.48,10.5]|low   |
|[5.0,1.04,0.24,1.6,0.05,32.0,96.0,0.9934,3.74,0.62,11.5]  |medium|
+-----------------------------------------------------------+------+
only showing top 5 rows
```

6. Fit Multinomial logisticRegression Classification Model

```
from pyspark.ml.classification import LogisticRegression
logr = LogisticRegression(featuresCol='indexedFeatures', labelCol='indexedLabel')
```

7. Pipeline Architecture

```
# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, logr,labelConverter])
```

```
# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

8. Make predictions

```
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
+--------------------+------+--------------+
|            features| label|predictedLabel|
+--------------------+------+--------------+
|[4.6,0.52,0.15,2....|   low|        medium|
|[4.9,0.42,0.0,2.1...|  high|          high|
|[5.0,0.42,0.24,2....|  high|          high|
|[5.0,1.02,0.04,1....|   low|        medium|
|[5.0,1.04,0.24,1....|medium|        medium|
+--------------------+------+--------------+
only showing top 5 rows
```

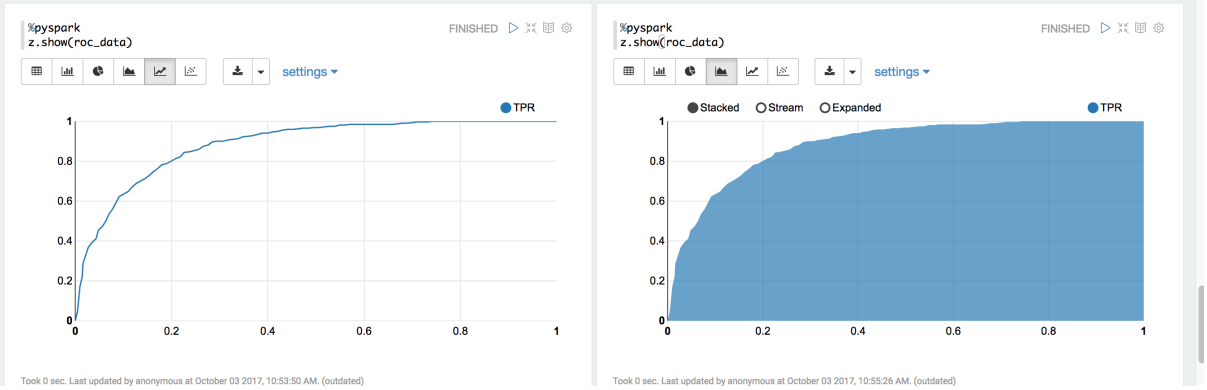
9. Evaluation

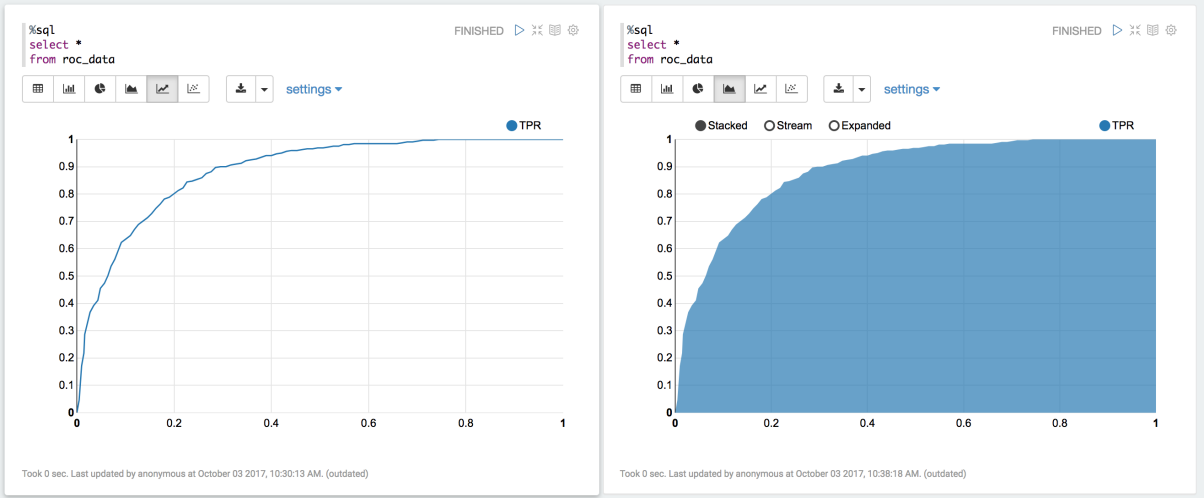
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))

Test Error = 0.181287

lrModel = model.stages[2]
trainingSummary = lrModel.summary

# Obtain the objective per iteration
# objectiveHistory = trainingSummary.objectiveHistory
# print("objectiveHistory:")
# for objective in objectiveHistory:
#     print(objective)

# Obtain the receiver-operating characteristic as a dataframe and areaUnderROC.
trainingSummary.roc.show(5)
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))

# Set the model threshold to maximize F-Measure
fMeasure = trainingSummary.fMeasureByThreshold
maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').head(5)
# bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-Measure)'])
#     .select('threshold').head()['threshold']
# lr.setThreshold(bestThreshold)
```

You can use `z.show()` to get the data and plot the ROC curves:

You can also register a TempTable `data.registerTempTable('roc_data')` and then use `sql` to plot the ROC curve:

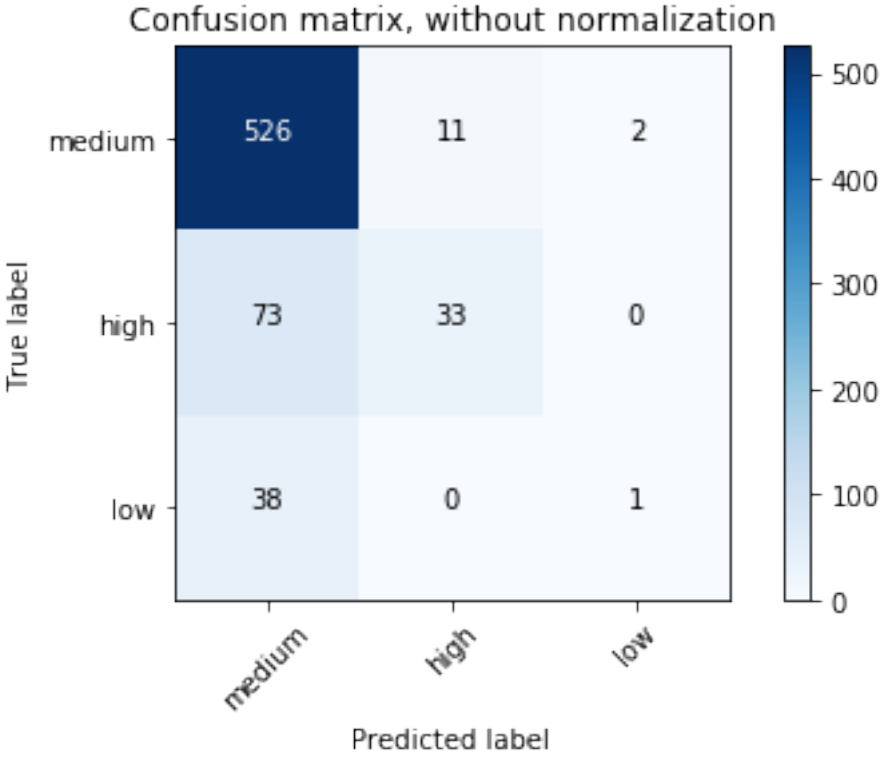
10. visualization

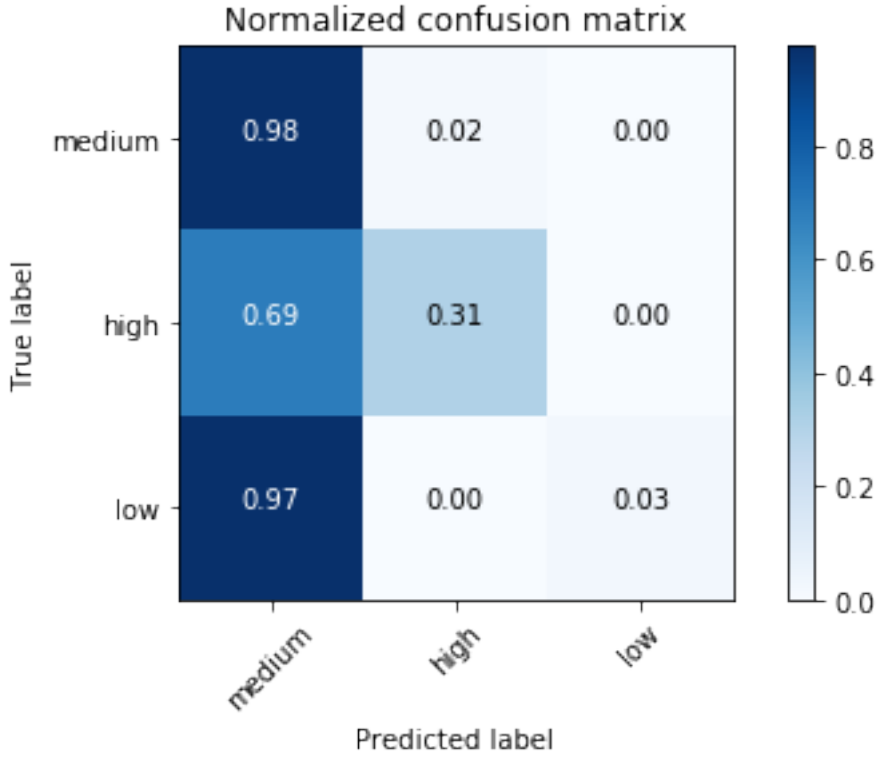





```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
```

```
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

class_temp = predictions.select("label").groupBy("label")\
                        .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names

['medium', 'high', 'low']

from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix

array([[526,  11,   2],
       [ 73,  33,   0],
       [ 38,   0,   1]])

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()

Confusion matrix, without normalization
[[526  11   2]
 [ 73  33   0]
 [ 38   0   1]]

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()

Normalized confusion matrix
[[0.97588126 0.02040816 0.00371058]
 [0.68867925 0.31132075 0.        ]
 [0.97435897 0.         0.02564103]]
```


Confusion matrix, without normalization
medium
high
low
True label
526
11
2
73
33
0
38
0
1
500
400
300
200
100
0
medium
high
low
Predicted label


Normalized confusion matrix
medium
high
low
True label
0.98
0.02
0.00
0.69
0.31
0.00
0.97
0.00
0.03
0.8
0.6
0.4
0.2
0.0
medium
high
low
Predicted label

10.3 Decision tree Classification

10.3.1 Introduction

10.3.2 Demo

- The Jupyter notebook can be download from Decision Tree Classification.
- For more details, please visit DecisionTreeClassifier API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

        spark = SparkSession \
            .builder \
            .appName("Python Spark Decision Tree classification") \
            .config("spark.some.config.option", "some-value") \
            .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                                options(header='true', \
                                inferschema='true') \
                .load("../data/WineData2.csv",header=True);
df.show(5,True)
```

```
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|fixed|volatile|citric|sugar|chlorides|free|total|density|  pH|sulphates|alcohol|quality
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4|      5
|  7.8|    0.88|   0.0|  2.6|    0.098|25.0| 67.0| 0.9968| 3.2|     0.68|    9.8|      5
|  7.8|    0.76|  0.04|  2.3|    0.092|15.0| 54.0|  0.997|3.26|     0.65|    9.8|      5
| 11.2|    0.28|  0.56|  1.9|    0.075|17.0| 60.0|  0.998|3.16|     0.58|    9.8|      6
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4|      5
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
only showing top 5 rows
```

```
# Convert to float format
def string_to_float(x):
    return float(x)

#
def condition(r):
    if (0<= r <= 4):
        label = "low"
    elif(4< r <= 6):
        label = "medium"
    else:
        label = "high"
    return label

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())
```

```
df = df.withColumn("quality", quality_udf("quality"))
df.show(5,True)
df.printSchema()
```

```
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|fixed|volatile|citric|sugar|chlorides|free|total|density|  pH|sulphates|alcohol|quality
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4| medium
|  7.8|    0.88|   0.0|  2.6|    0.098|25.0| 67.0| 0.9968| 3.2|     0.68|    9.8| medium
|  7.8|    0.76|  0.04|  2.3|    0.092|15.0| 54.0|  0.997|3.26|     0.65|    9.8| medium
| 11.2|    0.28|  0.56|  1.9|    0.075|17.0| 60.0|  0.998|3.16|     0.58|    9.8| medium
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4| medium
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
only showing top 5 rows

root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)
```

3. Convert the data to dense vector

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)
```

```
        data = data.withColumn('label',col(labelCol))

        return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
    def get_dummy(df,indexCol,categoricalCols,continuousCols):
        '''
        Get dummy variables and concat with continuous variables for unsupervised learn
        :param df: the dataframe
        :param categoricalCols: the name list of the categorical data
        :param continuousCols:  the name list of the numerical data
        :return k: feature matrix

        :author: Wenqiang Feng
        :email:  von198@gmail.com
        '''

        indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                     for c in categoricalCols ]

        # default setting: dropLast=True
        encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                     outputCol="{0}_encoded".format(indexer.getOutputCol()))
                     for indexer in indexers ]

        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
```

```
# !!!!caution: not from pyspark.mllib.linalg import Vectors
from pyspark.ml.linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString,StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label
```

4. Transform the dataset to DataFrame

```
transformed = transData(df)
transformed.show(5)
```

```
+--------------------+------+
|            features| label|
+--------------------+------+
|[7.4,0.7,0.0,1.9,...|medium|
|[7.8,0.88,0.0,2.6...|medium|
|[7.8,0.76,0.04,2....|medium|
```

```
|[11.2,0.28,0.56,1...|medium|
|[7.4,0.7,0.0,1.9,...|medium|
+--------------------+------+
only showing top 5 rows
```

5. Deal with Categorical Label and Variables

```
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+------------+
|            features| label|indexedLabel|
+--------------------+------+------------+
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
|[7.8,0.88,0.0,2.6...|medium|         0.0|
|[7.8,0.76,0.04,2....|medium|         0.0|
|[11.2,0.28,0.56,1...|medium|         0.0|
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
+--------------------+------+------------+
only showing top 5 rows
```

```
    # Automatically identify categorical features, and index them.
    # Set maxCategories so features with > 4 distinct values are treated as continuous.
    featureIndexer =VectorIndexer(inputCol="features", \
                                  outputCol="indexedFeatures", \
                                  maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+--------------------+
|            features| label|     indexedFeatures|
+--------------------+------+--------------------+
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
|[7.8,0.88,0.0,2.6...|medium|[7.8,0.88,0.0,2.6...|
|[7.8,0.76,0.04,2....|medium|[7.8,0.76,0.04,2....|
|[11.2,0.28,0.56,1...|medium|[11.2,0.28,0.56,1...|
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
+--------------------+------+--------------------+
only showing top 5 rows
```

6. Split the data to training and test data sets

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])

trainingData.show(5)
testData.show(5)
```

```
+--------------------+------+
|            features| label|
+--------------------+------+
|[4.6,0.52,0.15,2....|   low|
|[4.7,0.6,0.17,2.3...|medium|
|[5.0,1.02,0.04,1....|   low|
|[5.0,1.04,0.24,1....|medium|
|[5.1,0.585,0.0,1....|  high|
+--------------------+------+
```

```
only showing top 5 rows

+--------------------+------+
|            features| label|
+--------------------+------+
|[4.9,0.42,0.0,2.1...|  high|
|[5.0,0.38,0.01,1....|medium|
|[5.0,0.4,0.5,4.3,...|medium|
|[5.0,0.42,0.24,2....|  high|
|[5.0,0.74,0.0,1.2...|medium|
+--------------------+------+
only showing top 5 rows
```

7. Fit Decision Tree Classification Model

```
from pyspark.ml.classification import DecisionTreeClassifier

# Train a DecisionTree model
dTree = DecisionTreeClassifier(labelCol='indexedLabel', featuresCol='indexedFeatures')
```

8. Pipeline Architecture

```
# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dTree,labelConverter])

# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

9. Make predictions

```
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
+--------------------+------+--------------+
|            features| label|predictedLabel|
+--------------------+------+--------------+
|[4.9,0.42,0.0,2.1...|  high|          high|
|[5.0,0.38,0.01,1....|medium|        medium|
|[5.0,0.4,0.5,4.3,...|medium|        medium|
|[5.0,0.42,0.24,2....|  high|        medium|
|[5.0,0.74,0.0,1.2...|medium|        medium|
+--------------------+------+--------------+
only showing top 5 rows
```

10. Evaluation

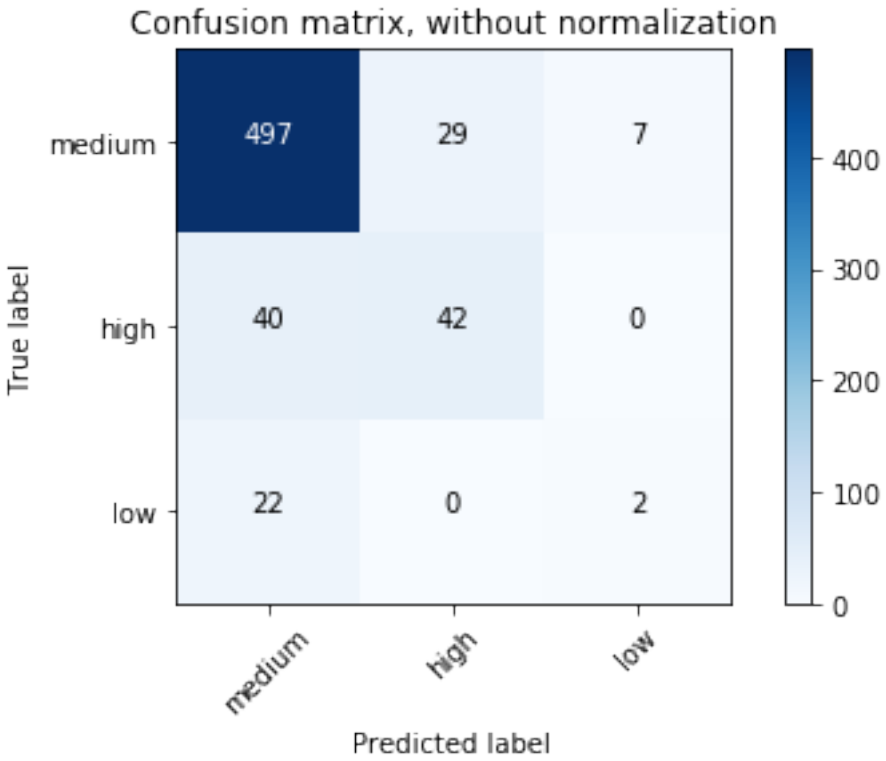
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
```

```
print("Test Error = %g" % (1.0 - accuracy))

rfModel = model.stages[-2]
print(rfModel)  # summary only

Test Error = 0.45509
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4545ac8dca9c8438ef2a)
of depth 5 with 59 nodes
```

11. visualization

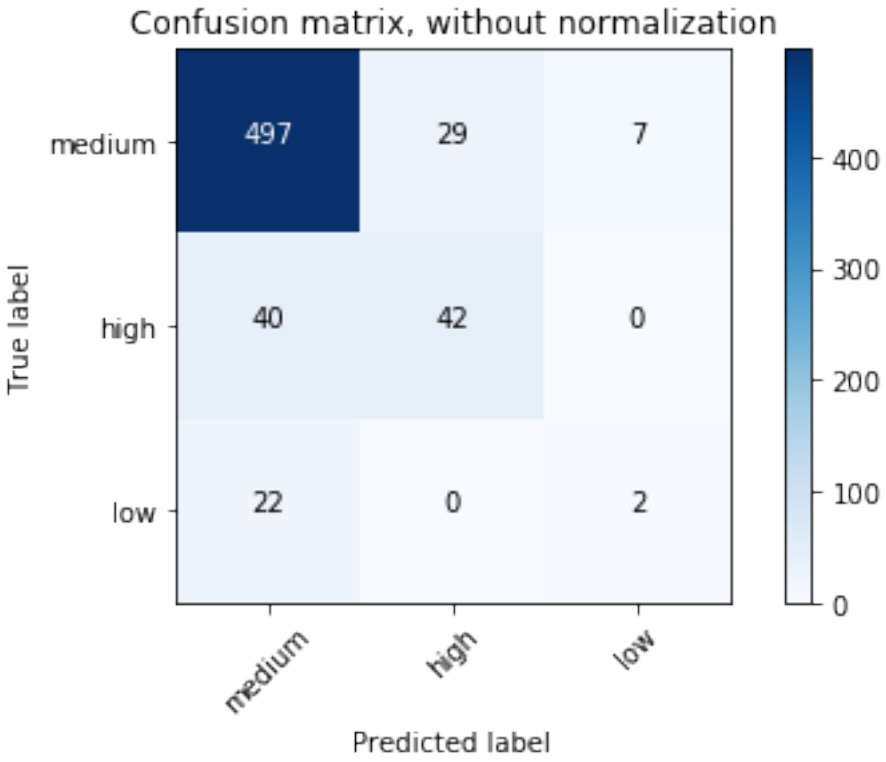
```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

class_temp = predictions.select("label").groupBy("label")\
                        .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names
```

```
['medium', 'high', 'low']

from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix

array([[497,  29,   7],
       [ 40,  42,   0],
       [ 22,   0,   2]])

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()

Confusion matrix, without normalization
[[497  29   7]
 [ 40  42   0]
 [ 22   0   2]]
```




```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')
```

```
plt.show()

Normalized confusion matrix
[[ 0.93245779  0.05440901  0.01313321]
 [ 0.48780488  0.51219512  0.        ]
 [ 0.91666667  0.          0.08333333]]
```



10.4 Random forest Classification

10.4.1 Introduction

10.4.2 Demo

- The Jupyter notebook can be download from Random forest Classification.
- For more details, please visit RandomForestClassifier API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

        spark = SparkSession \
            .builder \
            .appName("Python Spark Decision Tree classification") \
            .config("spark.some.config.option", "some-value") \
            .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                                options(header='true', \
                                inferschema='true') \
                .load("../data/WineData2.csv",header=True);
df.show(5,True)
```

```
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|fixed|volatile|citric|sugar|chlorides|free|total|density|  pH|sulphates|alcohol|quality
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4|      5
|  7.8|    0.88|   0.0|  2.6|    0.098|25.0| 67.0| 0.9968| 3.2|     0.68|    9.8|      5
|  7.8|    0.76|  0.04|  2.3|    0.092|15.0| 54.0|  0.997|3.26|     0.65|    9.8|      5
| 11.2|    0.28|  0.56|  1.9|    0.075|17.0| 60.0|  0.998|3.16|     0.58|    9.8|      6
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4|      5
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
only showing top 5 rows
```

```
# Convert to float format
def string_to_float(x):
    return float(x)

#
def condition(r):
    if (0<= r <= 4):
        label = "low"
    elif(4< r <= 6):
        label = "medium"
    else:
        label = "high"
    return label

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())

df = df.withColumn("quality", quality_udf("quality"))
df.show(5,True)
df.printSchema()
```

```
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|fixed|volatile|citric|sugar|chlorides|free|total|density|  pH|sulphates|alcohol|quality
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4| medium
|  7.8|    0.88|   0.0|  2.6|    0.098|25.0| 67.0| 0.9968| 3.2|     0.68|    9.8| medium
|  7.8|    0.76|  0.04|  2.3|    0.092|15.0| 54.0|  0.997|3.26|     0.65|    9.8| medium
| 11.2|    0.28|  0.56|  1.9|    0.075|17.0| 60.0|  0.998|3.16|     0.58|    9.8| medium
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4| medium
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
only showing top 5 rows

root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
```

```
|-- total: double (nullable = true)
|-- density: double (nullable = true)
|-- pH: double (nullable = true)
|-- sulphates: double (nullable = true)
|-- alcohol: double (nullable = true)
|-- quality: string (nullable = true)
```

3. Convert the data to dense vector

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for unsupervised learn
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
```

```
        encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                     outputCol="{0}_encoded".format(indexer.getOutputCol()))
                     for indexer in indexers ]

        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
```

```
# !!!!caution: not from pyspark.mllib.linalg import Vectors
from pyspark.ml.linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString,StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label
```

4. Transform the dataset to DataFrame

```
transformed = transData(df)
transformed.show(5)
```

```
+--------------------+------+
|            features| label|
+--------------------+------+
|[7.4,0.7,0.0,1.9,...|medium|
|[7.8,0.88,0.0,2.6...|medium|
|[7.8,0.76,0.04,2....|medium|
|[11.2,0.28,0.56,1...|medium|
|[7.4,0.7,0.0,1.9,...|medium|
+--------------------+------+
only showing top 5 rows
```

5. Deal with Categorical Label and Variables

```
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+------------+
|            features| label|indexedLabel|
+--------------------+------+------------+
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
|[7.8,0.88,0.0,2.6...|medium|         0.0|
|[7.8,0.76,0.04,2....|medium|         0.0|
|[11.2,0.28,0.56,1...|medium|         0.0|
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
+--------------------+------+------------+
only showing top 5 rows
```

```
    # Automatically identify categorical features, and index them.
    # Set maxCategories so features with > 4 distinct values are treated as continuous.
    featureIndexer =VectorIndexer(inputCol="features", \
                                  outputCol="indexedFeatures", \
                                  maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+--------------------+
|            features| label|     indexedFeatures|
+--------------------+------+--------------------+
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
|[7.8,0.88,0.0,2.6...|medium|[7.8,0.88,0.0,2.6...|
|[7.8,0.76,0.04,2....|medium|[7.8,0.76,0.04,2....|
|[11.2,0.28,0.56,1...|medium|[11.2,0.28,0.56,1...|
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
+--------------------+------+--------------------+
only showing top 5 rows
```

6. Split the data to training and test data sets

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])

trainingData.show(5)
testData.show(5)
```

```
+--------------------+------+
|            features| label|
+--------------------+------+
|[4.6,0.52,0.15,2....|   low|
|[4.7,0.6,0.17,2.3...|medium|
|[5.0,1.02,0.04,1....|   low|
|[5.0,1.04,0.24,1....|medium|
|[5.1,0.585,0.0,1....|  high|
+--------------------+------+
only showing top 5 rows

+--------------------+------+
|            features| label|
+--------------------+------+
|[4.9,0.42,0.0,2.1...|  high|
|[5.0,0.38,0.01,1....|medium|
|[5.0,0.4,0.5,4.3,...|medium|
|[5.0,0.42,0.24,2....|  high|
|[5.0,0.74,0.0,1.2...|medium|
+--------------------+------+
only showing top 5 rows
```

7. Fit Random Forest Classification Model

```
from pyspark.ml.classification import RandomForestClassifier

# Train a RandomForest model.
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numT
```

8. Pipeline Architecture

```
# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf,labelConverter])

# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

9. Make predictions

```
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
+--------------------+------+--------------+
|            features| label|predictedLabel|
+--------------------+------+--------------+
|[4.9,0.42,0.0,2.1...|  high|          high|
|[5.0,0.38,0.01,1....|medium|        medium|
|[5.0,0.4,0.5,4.3,...|medium|        medium|
|[5.0,0.42,0.24,2....|  high|        medium|
|[5.0,0.74,0.0,1.2...|medium|        medium|
+--------------------+------+--------------+
only showing top 5 rows
```

10. Evaluation

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))

rfModel = model.stages[-2]
print(rfModel)  # summary only
```

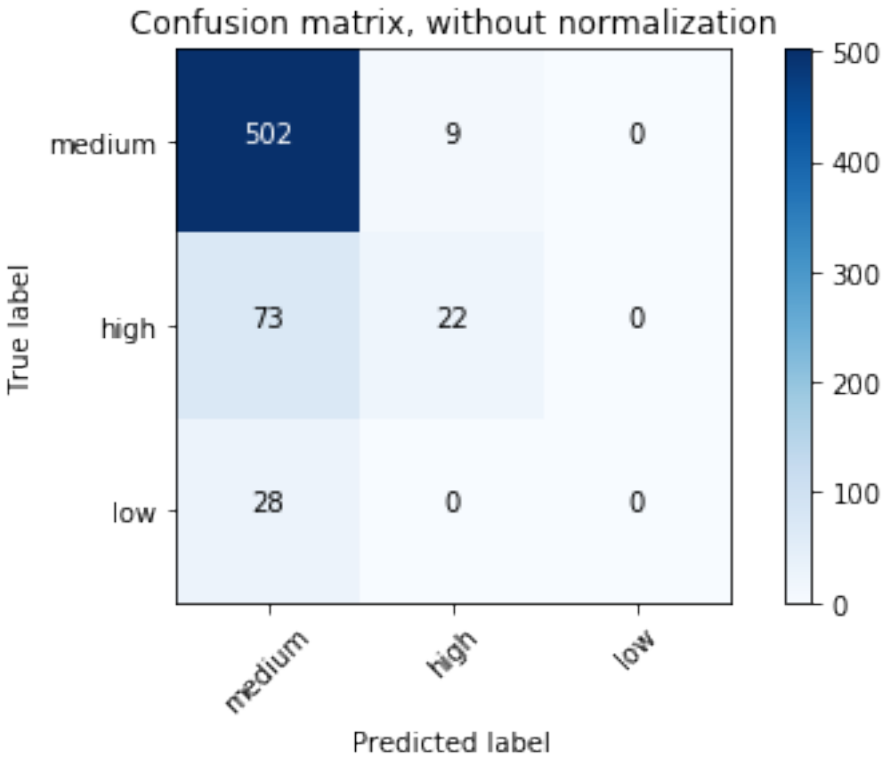
```
Test Error = 0.173502
RandomForestClassificationModel (uid=rfc_a3395531f1d2) with 10 trees
```

11. visualization

```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
```

```
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

class_temp = predictions.select("label").groupBy("label")\
                        .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names

['medium', 'high', 'low']

from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix

array([[502,   9,   0],
       [ 73,  22,   0],
       [ 28,   0,   0]])

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()

Confusion matrix, without normalization
[[502   9   0]
```

```
 [ 73  22   0]
 [ 28   0   0]]
```




```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()

Normalized confusion matrix
[[ 0.98238748  0.01761252  0.        ]
 [ 0.76842105  0.23157895  0.        ]
 [ 1.          0.          0.        ]]
```

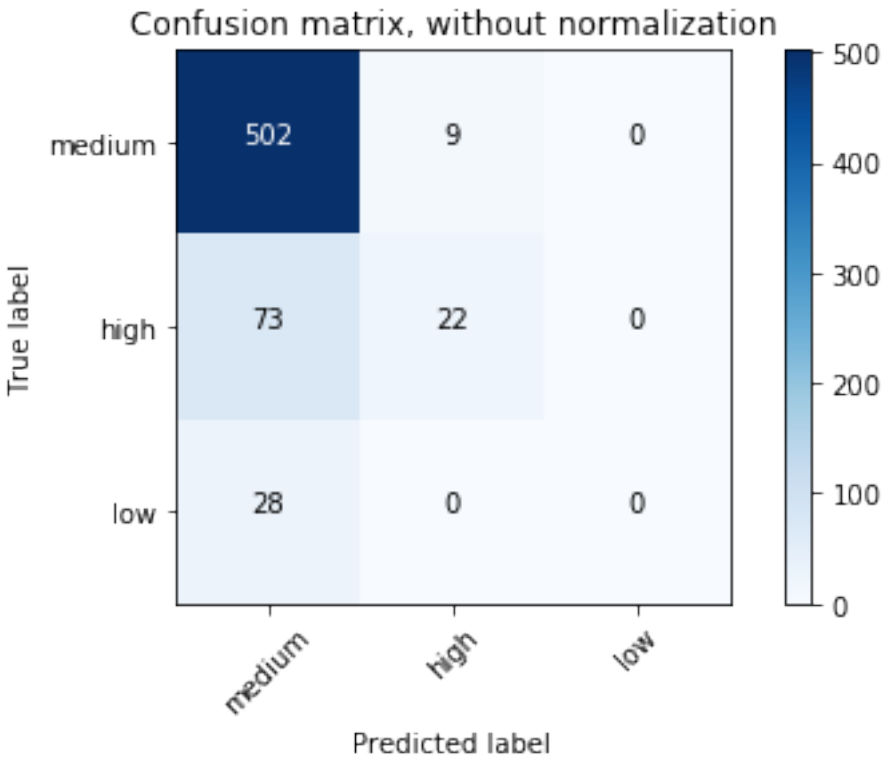
10.5 Gradient-boosted tree Classification

10.5.1 Introduction

10.5.2 Demo

- The Jupyter notebook can be download from Gradient boosted tree Classification.
- For more details, please visit GBTClassifier API .

Warning: Unfortunately, the GBTClassifier currently only supports binary labels.



10.6 Naive Bayes Classification

10.6.1 Introduction

10.6.2 Demo

- The Jupyter notebook can be download from Naive Bayes Classification.
- For more details, please visit NaiveBayes API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark  Naive Bayes classification") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv') \
            .options(header='true', inferschema='true') \
            .load("./data/WineData2.csv",header=True);
df.show(5)
```

```
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|fixed|volatile|citric|sugar|chlorides|free|total|density|  pH|sulphates|alcohol|quality
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4|      5
```

```
|  7.8|    0.88|   0.0|  2.6|    0.098|25.0| 67.0| 0.9968| 3.2|     0.68|    9.8|      5
|  7.8|    0.76|  0.04|  2.3|    0.092|15.0| 54.0|  0.997|3.26|     0.65|    9.8|      5
| 11.2|    0.28|  0.56|  1.9|    0.075|17.0| 60.0|  0.998|3.16|     0.58|    9.8|      6
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4|      5
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
only showing top 5 rows

df.printSchema()

root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)

# Convert to float format
def string_to_float(x):
    return float(x)

#
def condition(r):
    if (0<= r <= 6):
        label = "low"
    else:
        label = "high"
    return label

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())

df = df.withColumn("quality", quality_udf("quality"))

df.show(5,True)

+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|fixed|volatile|citric|sugar|chlorides|free|total|density|  pH|sulphates|alcohol|quality
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4| medium
|  7.8|    0.88|   0.0|  2.6|    0.098|25.0| 67.0| 0.9968| 3.2|     0.68|    9.8| medium
|  7.8|    0.76|  0.04|  2.3|    0.092|15.0| 54.0|  0.997|3.26|     0.65|    9.8| medium
| 11.2|    0.28|  0.56|  1.9|    0.075|17.0| 60.0|  0.998|3.16|     0.58|    9.8| medium
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4| medium
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
only showing top 5 rows

df.printSchema()
```

```
root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)
```

3. Deal with categorical data and Convert the data to dense vector

---

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for unsupervised learn
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
```

```
        :email:  von198@gmail.com
        '''

        indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                     for c in categoricalCols ]

        # default setting: dropLast=True
        encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                     outputCol="{0}_encoded".format(indexer.getOutputCol()))
                     for indexer in indexers ]

        assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in en
                                    + continuousCols, outputCol="features")

        pipeline = Pipeline(stages=indexers + encoders + [assembler])

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
```

```
def get_dummy(df,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in encoder
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select('features','label')
```

4. Transform the dataset to DataFrame

```
from pyspark.ml.linalg import Vectors # !!!!caution: not from pyspark.mllib.linalg impor
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString,StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

def transData(data):
return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label'])
```

```
transformed = transData(df)
transformed.show(5)
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[7.4,0.7,0.0,1.9,...|  low|
|[7.8,0.88,0.0,2.6...|  low|
|[7.8,0.76,0.04,2....|  low|
|[11.2,0.28,0.56,1...|  low|
|[7.4,0.7,0.0,1.9,...|  low|
+--------------------+-----+
only showing top 5 rows
```

4. Deal with Categorical Label and Variables

```
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+--------------------+-----+------------+
|            features|label|indexedLabel|
+--------------------+-----+------------+
|[7.4,0.7,0.0,1.9,...|  low|         0.0|
|[7.8,0.88,0.0,2.6...|  low|         0.0|
|[7.8,0.76,0.04,2....|  low|         0.0|
|[11.2,0.28,0.56,1...|  low|         0.0|
|[7.4,0.7,0.0,1.9,...|  low|         0.0|
+--------------------+-----+------------+
only showing top 5 rows
```

```
# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated as continuous.
featureIndexer =VectorIndexer(inputCol="features", \
                              outputCol="indexedFeatures", \
                              maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+--------------------+-----+--------------------+
|            features|label|     indexedFeatures|
+--------------------+-----+--------------------+
|[7.4,0.7,0.0,1.9,...|  low|[7.4,0.7,0.0,1.9,...|
|[7.8,0.88,0.0,2.6...|  low|[7.8,0.88,0.0,2.6...|
|[7.8,0.76,0.04,2....|  low|[7.8,0.76,0.04,2....|
|[11.2,0.28,0.56,1...|  low|[11.2,0.28,0.56,1...|
|[7.4,0.7,0.0,1.9,...|  low|[7.4,0.7,0.0,1.9,...|
+--------------------+-----+--------------------+
only showing top 5 rows
```

5. Split the data to training and test data sets

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5,False)
testData.show(5,False)
```

```
+-----------------------------------------------------------+-----+
|features                                                   |label|
+-----------------------------------------------------------+-----+
|[5.0,0.38,0.01,1.6,0.048,26.0,60.0,0.99084,3.7,0.75,14.0]|low  |
|[5.0,0.42,0.24,2.0,0.06,19.0,50.0,0.9917,3.72,0.74,14.0] |high |
|[5.0,0.74,0.0,1.2,0.041,16.0,46.0,0.99258,4.01,0.59,12.5]|low  |
|[5.0,1.02,0.04,1.4,0.045,41.0,85.0,0.9938,3.75,0.48,10.5]|low  |
|[5.0,1.04,0.24,1.6,0.05,32.0,96.0,0.9934,3.74,0.62,11.5] |low  |
+-----------------------------------------------------------+-----+
only showing top 5 rows

+-----------------------------------------------------------+-----+
|features                                                   |label|
+-----------------------------------------------------------+-----+
|[4.6,0.52,0.15,2.1,0.054,8.0,65.0,0.9934,3.9,0.56,13.1]   |low  |
|[4.7,0.6,0.17,2.3,0.058,17.0,106.0,0.9932,3.85,0.6,12.9]  |low  |
|[4.9,0.42,0.0,2.1,0.048,16.0,42.0,0.99154,3.71,0.74,14.0]|high |
|[5.0,0.4,0.5,4.3,0.046,29.0,80.0,0.9902,3.49,0.66,13.6]   |low  |
|[5.2,0.49,0.26,2.3,0.09,23.0,74.0,0.9953,3.71,0.62,12.2]  |low  |
+-----------------------------------------------------------+-----+
only showing top 5 rows
```

6. Fit Naive Bayes Classification Model

```
from pyspark.ml.classification import NaiveBayes
nb = NaiveBayes(featuresCol='indexedFeatures', labelCol='indexedLabel')
```

7. Pipeline Architecture

```
# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, nb,labelConverter])

# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

8. Make predictions

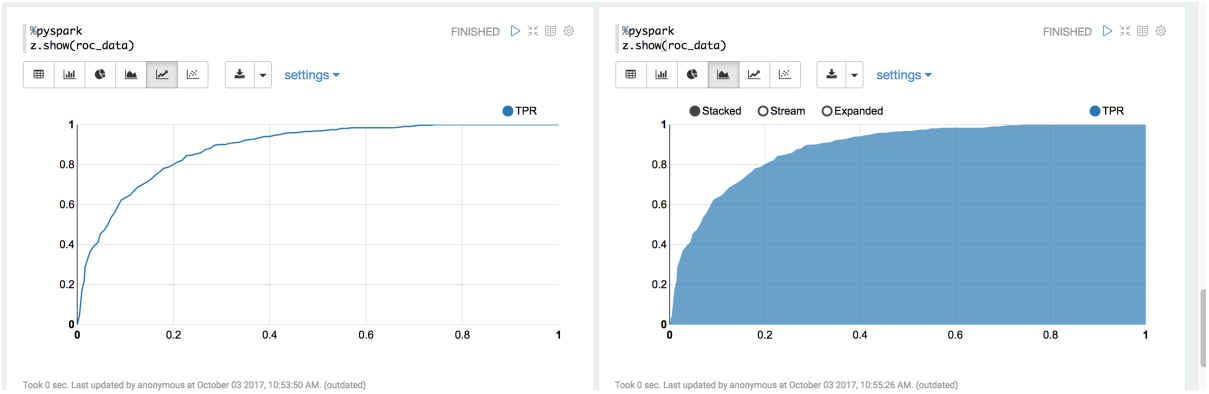
```
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
+--------------------+-----+--------------+
|            features|label|predictedLabel|
+--------------------+-----+--------------+
|[4.6,0.52,0.15,2....|  low|           low|
|[4.7,0.6,0.17,2.3...|  low|           low|
|[4.9,0.42,0.0,2.1...| high|           low|
|[5.0,0.4,0.5,4.3,...|  low|           low|
|[5.2,0.49,0.26,2....|  low|           low|
+--------------------+-----+--------------+
only showing top 5 rows
```

9. Evaluation

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))

Test Error = 0.307339

lrModel = model.stages[2]
trainingSummary = lrModel.summary

# Obtain the objective per iteration
# objectiveHistory = trainingSummary.objectiveHistory
# print("objectiveHistory:")
# for objective in objectiveHistory:
#     print(objective)

# Obtain the receiver-operating characteristic as a dataframe and areaUnderROC.
trainingSummary.roc.show(5)
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))

# Set the model threshold to maximize F-Measure
fMeasure = trainingSummary.fMeasureByThreshold
maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').head(5)
# bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-Measure)'])
#     .select('threshold').head()['threshold']
# lr.setThreshold(bestThreshold)
```

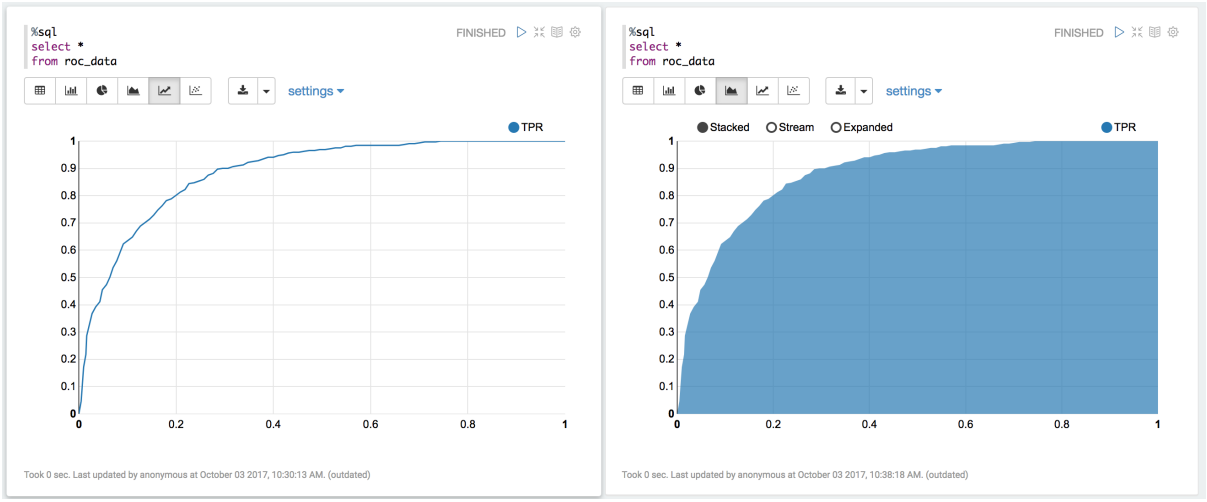
You can use `z.show()` to get the data and plot the ROC curves:



You can also register a TempTable `data.registerTempTable('roc_data')` and then use `sql` to plot the ROC curve:

10. visualization

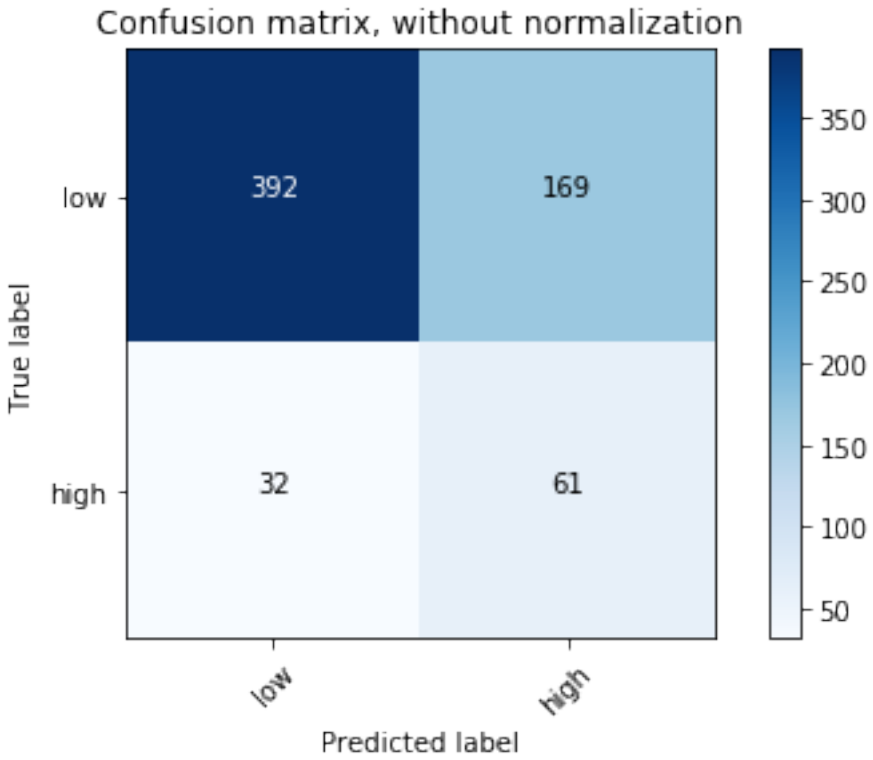
```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
```




```
                                cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

class_temp = predictions.select("label").groupBy("label")\
                        .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names
```

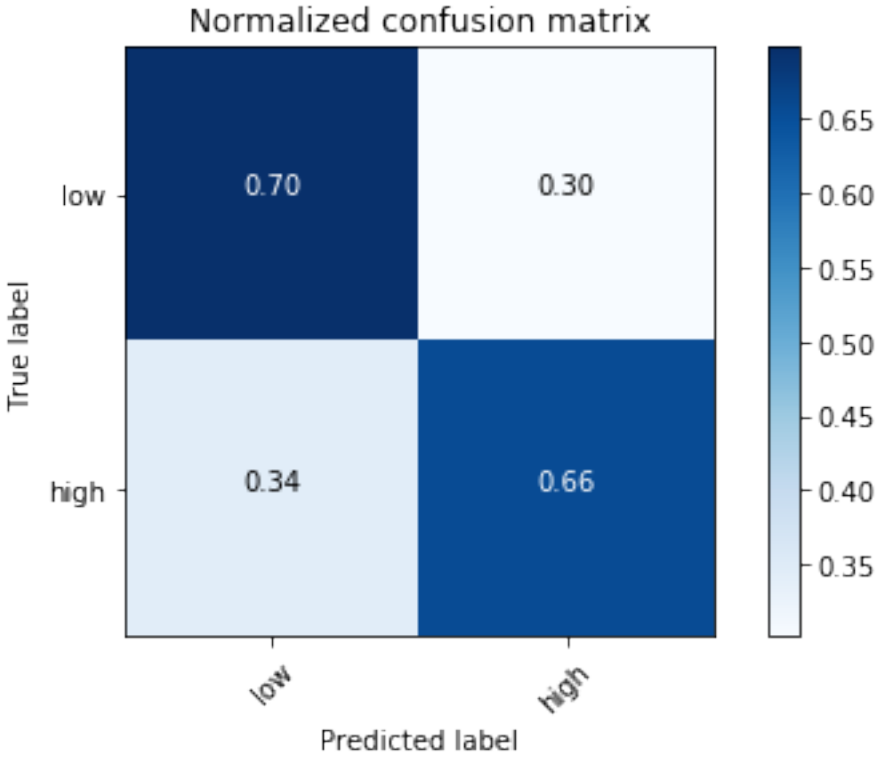
```
['low', 'high']
```

```
from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix

array([[392, 169],
       [ 32,  61]])

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()

Confusion matrix, without normalization
[[392 169]
 [ 32  61]]
```




```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[0.69875223 0.30124777]
 [0.34408602 0.65591398]]
```



CHAPTER

ELEVEN

CLUSTERING

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb

11.1 K-Means Model

11.1.1 Introduction

11.1.2 Demo

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark K-means example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                        options(header='true', \
                        inferschema='true').\
            load("../data/iris.csv",header=True);
```

check the data set

```
df.show(5,True)
df.printSchema()
```

Then you will get

```
+------------+-----------+------------+-----------+-------+
|sepal_length|sepal_width|petal_length|petal_width|species|
+------------+-----------+------------+-----------+-------+
|         5.1|        3.5|         1.4|        0.2| setosa|
|         4.9|        3.0|         1.4|        0.2| setosa|
|         4.7|        3.2|         1.3|        0.2| setosa|
|         4.6|        3.1|         1.5|        0.2| setosa|
|         5.0|        3.6|         1.4|        0.2| setosa|
+------------+-----------+------------+-----------+-------+
```

```
only showing top 5 rows

root
 |-- sepal_length: double (nullable = true)
 |-- sepal_width: double (nullable = true)
 |-- petal_length: double (nullable = true)
 |-- petal_width: double (nullable = true)
 |-- species: string (nullable = true)
```

You can also get the Statistical resutls from the data frame (Unfortunately, it only works for numerical).

```
df.describe().show()
```

Then you will get

```
+-------+------------------+-------------------+------------------+-------------------+--
|summary|      sepal_length|        sepal_width|      petal_length|        petal_width|
+-------+------------------+-------------------+------------------+-------------------+--
|  count|               150|                150|               150|                150|
|   mean| 5.843333333333335| 3.0540000000000007|3.7586666666666693|1.1986666666666672|
| stddev|0.8280661279778637|0.43359431136217375| 1.764420419952262|0.7631607417008414|
|    min|               4.3|                2.0|               1.0|                0.1|
|    max|               7.9|                4.4|               6.9|                2.5|vi
+-------+------------------+-------------------+------------------+-------------------+--
```

3. Convert the data to dense vector (**features**)

```
# convert the data to dense vector
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1])]).toDF(['features'])
```

4. Transform the dataset to DataFrame

```
transformed= transData(df)
transformed.show(5, False)
```

```
+-----------------+
|features         |
+-----------------+
|[5.1,3.5,1.4,0.2]|
|[4.9,3.0,1.4,0.2]|
|[4.7,3.2,1.3,0.2]|
|[4.6,3.1,1.5,0.2]|
|[5.0,3.6,1.4,0.2]|
+-----------------+
only showing top 5 rows
```

5. Deal With Categorical Variables

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4 distinct values are treated as continuou

featureIndexer = VectorIndexer(inputCol="features", \
                               outputCol="indexedFeatures",\
```

```
                                    maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
```

Now you check your dataset with

```
data.show(5,True)
```

you will get

```
+-----------------+-----------------+
|         features|  indexedFeatures|
+-----------------+-----------------+
|[5.1,3.5,1.4,0.2]|[5.1,3.5,1.4,0.2]|
|[4.9,3.0,1.4,0.2]|[4.9,3.0,1.4,0.2]|
|[4.7,3.2,1.3,0.2]|[4.7,3.2,1.3,0.2]|
|[4.6,3.1,1.5,0.2]|[4.6,3.1,1.5,0.2]|
|[5.0,3.6,1.4,0.2]|[5.0,3.6,1.4,0.2]|
+-----------------+-----------------+
only showing top 5 rows
```

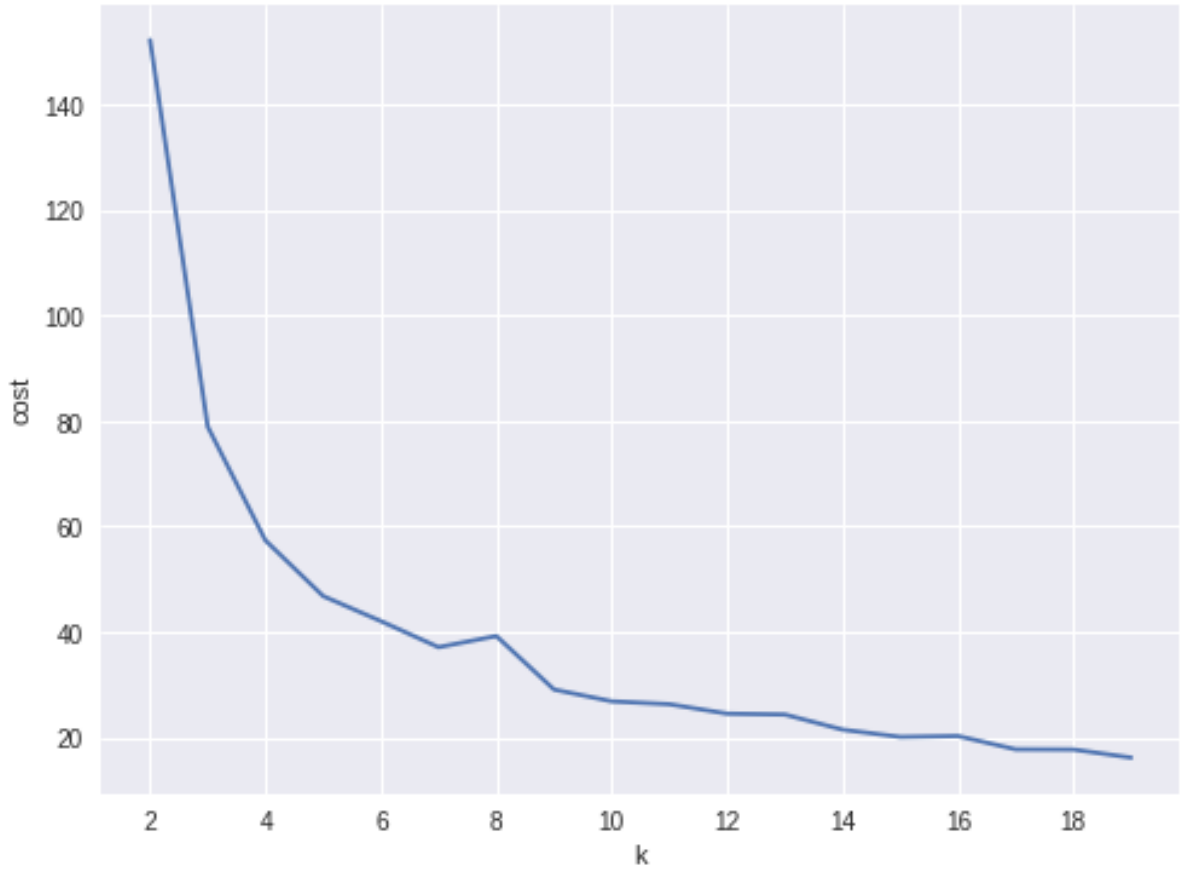
6. Elbow method to determine the optimal number of clusters for k-means clustering

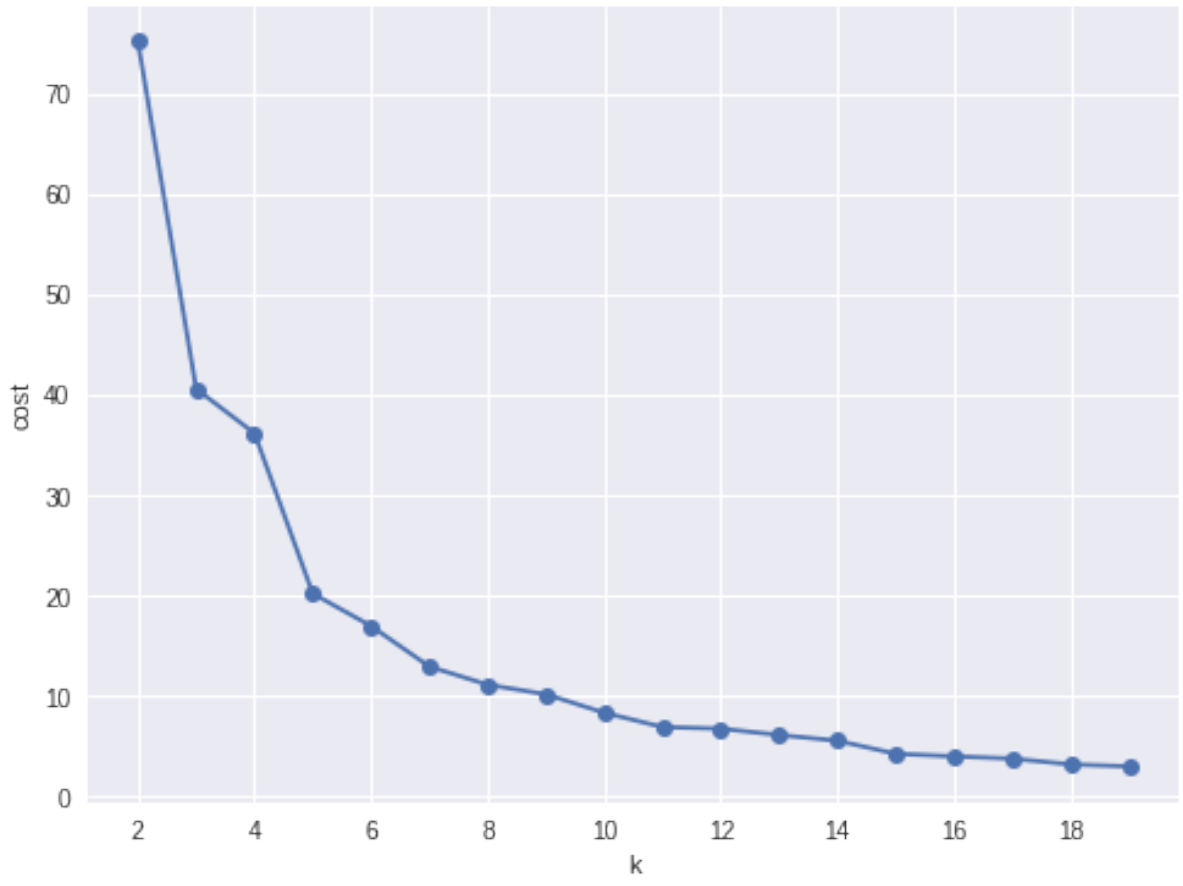
```
import numpy as np
cost = np.zeros(20)
for k in range(2,20):
    kmeans = KMeans()\
             .setK(k)\
             .setSeed(1) \
             .setFeaturesCol("indexedFeatures")\
             .setPredictionCol("cluster")

    model = kmeans.fit(data)
    cost[k] = model.computeCost(data) # requires Spark 2.0 or later

import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import seaborn as sbs
from matplotlib.ticker import MaxNLocator

fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,20),cost[2:20])
ax.set_xlabel('k')
ax.set_ylabel('cost')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```

In my opinion, sometimes it's hard to choose the optimal number of the clusters by using the `elbow method`. As shown in the following Figure, you can choose 3, 5 or even 8. I will choose `3` in this demo.

- Silhouette analysis

```
#PySpark libraries
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import col, percent_rank, lit
```


cost
140
120
100
80
60
40
20
2
4
6
8
10
12
14
16
18
k


cost
70
60
50
40
30
20
10
0
2
4
6
8
10
12
14
16
18
k

```
from pyspark.sql.window import Window
from pyspark.sql import DataFrame, Row
from pyspark.sql.types import StructType
from functools import reduce  # For Python 3.x

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

def optimal_k(df_in,index_col,k_min, k_max,num_runs):
    '''
    Determine optimal number of clusters by using Silhoutte Score Analysis.
    :param df_in: the input dataframe
    :param index_col: the name of the index column
    :param k_min: the train dataset
    :param k_min: the minmum number of the clusters
    :param k_max: the maxmum number of the clusters
    :param num_runs: the number of runs for each fixed clusters

    :return k: optimal number of the clusters
    :return silh_lst: Silhouette score
    :return r_table: the running results table

    :author: Wenqiang Feng
    :email:  WFeng@dstsystems.com
    '''

    start = time.time()
    silh_lst = []
    k_lst = np.arange(k_min, k_max+1)

    r_table = df_in.select(index_col).toPandas()
    r_table = r_table.set_index(index_col)
    centers = pd.DataFrame()

    for k in k_lst:
        silh_val = []
        for run in np.arange(1, num_runs+1):

            # Trains a k-means model.
            kmeans = KMeans()\
                    .setK(k)\
                    .setSeed(int(np.random.randint(100, size=1)))
            model = kmeans.fit(df_in)

            # Make predictions
            predictions = model.transform(df_in)
            r_table['cluster_{k}_{run}'.format(k=k, run=run)]= predictions.select('predi

            # Evaluate clustering by computing Silhouette score
            evaluator = ClusteringEvaluator()
            silhouette = evaluator.evaluate(predictions)
            silh_val.append(silhouette)

        silh_array=np.asanyarray(silh_val)
        silh_lst.append(silh_array.mean())

    elapsed =  time.time() - start
```

```
    silhouette = pd.DataFrame(list(zip(k_lst,silh_lst)),columns = ['k', 'silhouette'])

    print('+-------------------------------------------------------------+')
    print("|         The finding optimal k phase took %8.0f s.       |" %(elapsed))
    print('+-------------------------------------------------------------+')


    return k_lst[np.argmax(silh_lst, axis=0)], silhouette , r_table

k, silh_lst, r_table = optimal_k(scaledData,index_col,k_min, k_max,num_runs)

+-------------------------------------------------------------+
|         The finding optimal k phase took     1783 s.       |
+-------------------------------------------------------------+

spark.createDataFrame(silh_lst).show()

+---+------------------+
|  k|        silhouette|
+---+------------------+
|  3|0.8045154385557953|
|  4|0.6993528775512052|
|  5|0.6689286654221447|
|  6|0.6356184024841809|
|  7|0.7174102265711756|
|  8|0.6720861758298997|
|  9| 0.601771359881241|
| 10|0.6292447334578428|
+---+------------------+
```

From the silhouette list, we can choose 3 as the optimal number of the clusters.

Warning: `ClusteringEvaluator` in `pyspark.ml.evaluation` requires Spark 2.4 or later!!

7. Pipeline Architecture

```
from pyspark.ml.clustering import KMeans, KMeansModel

kmeans = KMeans() \
          .setK(3) \
          .setFeaturesCol("indexedFeatures")\
          .setPredictionCol("cluster")

# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, kmeans])

model = pipeline.fit(transformed)

cluster = model.transform(transformed)
```

8. k-means clusters

```
cluster = model.transform(transformed)

+-----------------+-----------------+-------+
|         features|  indexedFeatures|cluster|
+-----------------+-----------------+-------+
```

```
|[5.1,3.5,1.4,0.2]|[5.1,3.5,1.4,0.2]|      1|
|[4.9,3.0,1.4,0.2]|[4.9,3.0,1.4,0.2]|      1|
|[4.7,3.2,1.3,0.2]|[4.7,3.2,1.3,0.2]|      1|
|[4.6,3.1,1.5,0.2]|[4.6,3.1,1.5,0.2]|      1|
|[5.0,3.6,1.4,0.2]|[5.0,3.6,1.4,0.2]|      1|
|[5.4,3.9,1.7,0.4]|[5.4,3.9,1.7,0.4]|      1|
|[4.6,3.4,1.4,0.3]|[4.6,3.4,1.4,0.3]|      1|
|[5.0,3.4,1.5,0.2]|[5.0,3.4,1.5,0.2]|      1|
|[4.4,2.9,1.4,0.2]|[4.4,2.9,1.4,0.2]|      1|
|[4.9,3.1,1.5,0.1]|[4.9,3.1,1.5,0.1]|      1|
|[5.4,3.7,1.5,0.2]|[5.4,3.7,1.5,0.2]|      1|
|[4.8,3.4,1.6,0.2]|[4.8,3.4,1.6,0.2]|      1|
|[4.8,3.0,1.4,0.1]|[4.8,3.0,1.4,0.1]|      1|
|[4.3,3.0,1.1,0.1]|[4.3,3.0,1.1,0.1]|      1|
|[5.8,4.0,1.2,0.2]|[5.8,4.0,1.2,0.2]|      1|
|[5.7,4.4,1.5,0.4]|[5.7,4.4,1.5,0.4]|      1|
|[5.4,3.9,1.3,0.4]|[5.4,3.9,1.3,0.4]|      1|
|[5.1,3.5,1.4,0.3]|[5.1,3.5,1.4,0.3]|      1|
|[5.7,3.8,1.7,0.3]|[5.7,3.8,1.7,0.3]|      1|
|[5.1,3.8,1.5,0.3]|[5.1,3.8,1.5,0.3]|      1|
+-----------------+-----------------+-------+
only showing top 20 rows
```

CHAPTER

TWELVE

RFM ANALYSIS

Segment	RFM	Description	Marketing
Best Customers	111	Bought most recently and most often, and spend the most	No price incentives, new products, and loyalty programs
Loyal Customers	X1X	Buy most frequently	Use R and M to further segment
Big Spenders	XX1	Spend the most	Market your most expensive products
Almost Lost	311	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Customers	411	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Cheap Customers	444	Last purchased long ago, purchased few, and spent little	Don't spend too much trying to re-acquire

The above figure source: Blast Analytics Marketing

RFM is a method used for analyzing customer value. It is commonly used in database marketing and direct marketing and has received particular attention in retail and professional services industries. More details can be found at Wikipedia RFM_wikipedia.

RFM stands for the three dimensions:

- Recency – How recently did the customer purchase? i.e. Duration since last purchase
- Frequency – How often do they purchase? i.e. Total number of purchases

- Monetary Value – How much do they spend? i.e. Total money this customer spent

12.1 RFM Analysis Methodology

RFM Analysis contains three main steps:

12.1.1 1. Build the RFM features matrix for each customer

```
+----------+-------+---------+---------+
|CustomerID|Recency|Frequency| Monetary|
+----------+-------+---------+---------+
|     14911|      1|      248|132572.62|
|     12748|      0|      224|  29072.1|
|     17841|      1|      169| 40340.78|
|     14606|      1|      128| 11713.85|
|     15311|      0|      118| 59419.34|
+----------+-------+---------+---------+
only showing top 5 rows
```

12.1.2 2. Determine cutting points for each feature

```
+----------+-------+---------+--------+-----+-----+-----+
|CustomerID|Recency|Frequency|Monetary|r_seg|f_seg|m_seg|
+----------+-------+---------+--------+-----+-----+-----+
|     17420|     50|        3|  598.83|    2|    3|    2|
|     16861|     59|        3|  151.65|    3|    3|    1|
|     16503|    106|        5| 1421.43|    3|    2|    3|
|     15727|     16|        7| 5178.96|    1|    1|    4|
|     17389|      0|       43|31300.08|    1|    1|    4|
+----------+-------+---------+--------+-----+-----+-----+
only showing top 5 rows
```

12.1.3 3. Determine the RFM scores and summarize the corresponding business value

```
+----------+-------+---------+--------+-----+-----+-----+--------+
|CustomerID|Recency|Frequency|Monetary|r_seg|f_seg|m_seg|RFMScore|
+----------+-------+---------+--------+-----+-----+-----+--------+
|     17988|     11|        8|  191.17|    1|    1|    1|     111|
|     16892|      1|        7|  496.84|    1|    1|    2|     112|
|     16668|     15|        6|  306.72|    1|    1|    2|     112|
|     16554|      3|        7|  641.55|    1|    1|    2|     112|
|     16500|      4|        6|  400.86|    1|    1|    2|     112|
+----------+-------+---------+--------+-----+-----+-----+--------+
only showing top 5 rows
```

The corresponding business description and marketing value:

Segment	RFM	Description	Marketing
Best Customers	111	Bought most recently and most often, and spend the most	No price incentives, new products, and loyalty programs
Loyal Customers	X1X	Buy most frequently	Use R and M to further segment
Big Spenders	XX1	Spend the most	Market your most expensive products
Almost Lost	311	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Customers	411	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Cheap Customers	444	Last purchased long ago, purchased few, and spent little	Don't spend too much trying to re-acquire

Figure 12.1: Source: Blast Analytics Marketing

12.2 Demo

- The Jupyter notebook can be download from Data Exploration.
- The data can be downloaf from German Credit.

12.2.1 Load and clean data

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark RFM example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df_raw = spark.read.format('com.databricks.spark.csv').\
                        options(header='true', \
                        inferschema='true').\
            load("Online Retail.csv",header=True);
```

check the data set

```
df_raw.show(5)
df_raw.printSchema()
```

Then you will get

```
+---------+---------+--------------------+--------+-----------+---------+----------+---
|InvoiceNo|StockCode|         Description|Quantity| InvoiceDate|UnitPrice|CustomerID|
+---------+---------+--------------------+--------+-----------+---------+----------+---
|   536365|   85123A|WHITE HANGING HEA...|       6|12/1/10 8:26|     2.55|     17850|Uni
|   536365|    71053| WHITE METAL LANTERN|       6|12/1/10 8:26|     3.39|     17850|Uni
|   536365|   84406B|CREAM CUPID HEART...|       8|12/1/10 8:26|     2.75|     17850|Uni
|   536365|   84029G|KNITTED UNION FLA...|       6|12/1/10 8:26|     3.39|     17850|Uni
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|12/1/10 8:26|     3.39|     17850|Uni
+---------+---------+--------------------+--------+-----------+---------+----------+---
only showing top 5 rows

root
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: string (nullable = true)
 |-- UnitPrice: double (nullable = true)
 |-- CustomerID: integer (nullable = true)
 |-- Country: string (nullable = true)
```

3. Data clean and data manipulation

- check and remove the `null` values

```
from pyspark.sql.functions import count

def my_count(df_in):
    df_in.agg( *[ count(c).alias(c) for c in df_in.columns ] ).show()

my_count(df_raw)
```

```
+---------+---------+-----------+--------+-----------+---------+----------+-------+
|InvoiceNo|StockCode|Description|Quantity|InvoiceDate|UnitPrice|CustomerID|Country|
+---------+---------+-----------+--------+-----------+---------+----------+-------+
|   541909|   541909|     540455|  541909|     541909|   541909|    406829| 541909|
+---------+---------+-----------+--------+-----------+---------+----------+-------+
```

Since the count results are not the same, we have some null value in the `CustomerID` column. We can drop these records from the dataset.

```
df = df_raw.dropna(how='any')
my_count(df)
```

```
+---------+---------+-----------+--------+-----------+---------+----------+-------+
|InvoiceNo|StockCode|Description|Quantity|InvoiceDate|UnitPrice|CustomerID|Country|
+---------+---------+-----------+--------+-----------+---------+----------+-------+
|   406829|   406829|     406829|  406829|     406829|   406829|    406829| 406829|
+---------+---------+-----------+--------+-----------+---------+----------+-------+
```

- Dealwith the InvoiceDate

```
from pyspark.sql.functions import to_utc_timestamp, unix_timestamp, lit, datediff, col

timeFmt = "MM/dd/yy HH:mm"

df = df.withColumn('NewInvoiceDate'
                   , to_utc_timestamp(unix_timestamp(col('InvoiceDate'),timeFmt).cast('tim
                   , 'UTC'))

df.show(5)
```

```
+---------+---------+--------------------+--------+--------------+---------+----------+---
|InvoiceNo|StockCode|         Description|Quantity|   InvoiceDate|UnitPrice|CustomerID|
+---------+---------+--------------------+--------+--------------+---------+----------+---
|   536365|   85123A|WHITE HANGING HEA...|       6|12/1/10 8:26|     2.55|     17850|Uni
|   536365|    71053| WHITE METAL LANTERN|       6|12/1/10 8:26|     3.39|     17850|Uni
|   536365|   84406B|CREAM CUPID HEART...|       8|12/1/10 8:26|     2.75|     17850|Uni
|   536365|   84029G|KNITTED UNION FLA...|       6|12/1/10 8:26|     3.39|     17850|Uni
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|12/1/10 8:26|     3.39|     17850|Uni
+---------+---------+--------------------+--------+--------------+---------+----------+---
only showing top 5 rows
```

> **Warning:** The spark is pretty sensitive to the date format!

- calculate total price

```
from pyspark.sql.functions import round

df = df.withColumn('TotalPrice', round( df.Quantity * df.UnitPrice, 2 ) )
```

- calculate the time difference

```
from pyspark.sql.functions import mean, min, max, sum, datediff, to_date

date_max = df.select(max('NewInvoiceDate')).toPandas()
current = to_utc_timestamp( unix_timestamp(lit(str(date_max.iloc[0][0])), \
                                'yy-MM-dd HH:mm').cast('timestamp'), 'UTC' )

# Calculatre Duration
df = df.withColumn('Duration', datediff(lit(current), 'NewInvoiceDate'))
```

- build the Recency, Frequency and Monetary

```
recency = df.groupBy('CustomerID').agg(min('Duration').alias('Recency'))
frequency = df.groupBy('CustomerID', 'InvoiceNo').count()\
                        .groupBy('CustomerID')\
                        .agg(count("*").alias("Frequency"))
monetary = df.groupBy('CustomerID').agg(round(sum('TotalPrice'), 2).alias('Monetary'))
rfm = recency.join(frequency,'CustomerID', how = 'inner')\
             .join(monetary,'CustomerID', how = 'inner')

rfm.show(5)
```

```
+----------+-------+---------+--------+
|CustomerID|Recency|Frequency|Monetary|
+----------+-------+---------+--------+
|     17420|     50|        3|  598.83|
|     16861|     59|        3|  151.65|
|     16503|    106|        5| 1421.43|
|     15727|     16|        7| 5178.96|
|     17389|      0|       43|31300.08|
+----------+-------+---------+--------+
only showing top 5 rows
```

12.2.2 RFM Segmentation

4. Determine cutting points

In this section, you can use the techniques (statistical results and visualizations) in *Data Exploration* section to help you determine the cutting points for each attribute. In my opinion, the cutting points are mainly depend on the business sense. You's better talk to your makrting people and get feedback and suggestion from them. I will use the quantile as the cutting points in this demo.

```
cols = ['Recency','Frequency','Monetary']
describe_pd(rfm,cols,1)
```

```
+-------+-----------------+-----------------+------------------+
|summary|          Recency|        Frequency|          Monetary|
+-------+-----------------+-----------------+------------------+
|  count|           4372.0|           4372.0|            4372.0|
|   mean|91.58119853613907| 5.07548032936871|1898.4597003659655|
| stddev|100.7721393138483|9.338754163574727| 8219.345141139722|
|    min|              0.0|              1.0|          -4287.63|
|    max|            373.0|            248.0|         279489.02|
|    25%|             16.0|              1.0|293.36249999999995|
|    50%|             50.0|              3.0|           648.075|
|    75%|            143.0|              5.0|          1611.725|
+-------+-----------------+-----------------+------------------+
```

The user defined function by using the cutting points:

```
def RScore(x):
    if  x <= 16:
        return 1
    elif x<= 50:
        return 2
    elif x<= 143:
        return 3
    else:
        return 4

def FScore(x):
    if  x <= 1:
        return 4
    elif x <= 3:
        return 3
    elif x <= 5:
        return 2
    else:
        return 1

def MScore(x):
    if  x <= 293:
        return 4
    elif x <= 648:
        return 3
    elif x <= 1611:
        return 2
    else:
        return 1

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType

R_udf = udf(lambda x: RScore(x), StringType())
F_udf = udf(lambda x: FScore(x), StringType())
M_udf = udf(lambda x: MScore(x), StringType())
```

5. RFM Segmentation

```
rfm_seg = rfm.withColumn("r_seg", R_udf("Recency"))
rfm_seg = rfm_seg.withColumn("f_seg", F_udf("Frequency"))
rfm_seg = rfm_seg.withColumn("m_seg", M_udf("Monetary"))
rfm_seg = rfm_seg.withColumn('RFMScore',
                             F.concat(F.col('r_seg'),F.col('f_seg'), F.col('m_seg')))
rfm_seg.sort(F.col('RFMScore')).show(5)
```

12.2.3 Statistical Summary

6. Statistical Summary

- simple summary

```
rfm_seg.groupBy('RFMScore')\
       .agg({'Recency':'mean',
             'Frequency': 'mean',
```

```
            'Monetary': 'mean'} )\
        .sort(F.col('RFMScore')).show(5)
```

```
+--------+-----------------+------------------+------------------+
|RFMScore|     avg(Recency)|     avg(Monetary)|    avg(Frequency)|
+--------+-----------------+------------------+------------------+
|     111|             11.0|            191.17|               8.0|
|     112|              8.0|          505.9775|               7.5|
|     113|7.237113402061856|1223.3604123711339| 7.752577319587629|
|     114|6.035123966942149| 8828.888595041324|18.882231404958677|
|     121|              9.6|            207.24|               4.4|
+--------+-----------------+------------------+------------------+
only showing top 5 rows
```

- complex summary

```
grp = 'RFMScore'
num_cols = ['Recency','Frequency','Monetary']
df_input = rfm_seg

quantile_grouped = quantile_agg(df_input,grp,num_cols)
quantile_grouped.toPandas().to_csv(output_dir+'quantile_grouped.csv')

deciles_grouped = deciles_agg(df_input,grp,num_cols)
deciles_grouped.toPandas().to_csv(output_dir+'deciles_grouped.csv')
```

12.3 Extension

You can also apply the K-means clustering in *Clustering* section to do the segmentation.

12.3.1 Build feature matrix

1. build dense feature matrix

```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# method 1 (good for small feature):
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
#                                       row["Radio"],
#                                       row["Newspaper"]]))

# Method 2 (good for large features):
def transData(data):
    return data.rdd.map(lambda r: [r[0],Vectors.dense(r[1:])]).toDF(['CustomerID','rfm']

transformed= transData(rfm)
transformed.show(5)
```

```
+----------+------------------+
|CustomerID|               rfm|
+----------+------------------+
```

```
|     17420|  [50.0,3.0,598.83]|
|     16861|  [59.0,3.0,151.65]|
|     16503|[106.0,5.0,1421.43]|
|     15727| [16.0,7.0,5178.96]|
|     17389|[0.0,43.0,31300.08]|
+----------+-------------------+
only showing top 5 rows
```

2. Scaler the feature matrix

```
from pyspark.ml.feature import MinMaxScaler

scaler = MinMaxScaler(inputCol="rfm",\
         outputCol="features")
scalerModel =  scaler.fit(transformed)
scaledData = scalerModel.transform(transformed)
scaledData.show(5,False)
```

```
+----------+-------------------+-------------------------------------------------------------
|CustomerID|rfm                |features
+----------+-------------------+-------------------------------------------------------------
|17420     |[50.0,3.0,598.83]  |[0.13404825737265416,0.008097165991902834,0.017219387148
|16861     |[59.0,3.0,151.65]  |[0.1581769436997319,0.008097165991902834,0.015643570392
|16503     |[106.0,5.0,1421.43]|[0.28418230563002683,0.016194331983805668,0.020118145731
|15727     |[16.0,7.0,5178.96] |[0.04289544235924933,0.024291497975708502,0.033359298589
|17389     |[0.0,43.0,31300.08]|[0.0,0.1700404858299595,0.12540746393334334]
+----------+-------------------+-------------------------------------------------------------
only showing top 5 rows
```

12.3.2 K-means clustering

3. Find optimal number of cluster

I will present two popular ways to determine the optimal number of the cluster.

- elbow analysis

```
#PySpark libraries
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import col, percent_rank, lit
from pyspark.sql.window import Window
from pyspark.sql import DataFrame, Row
from pyspark.sql.types import StructType
from functools import reduce  # For Python 3.x

from pyspark.ml.clustering import KMeans
#from pyspark.ml.evaluation import ClusteringEvaluator  # requires Spark 2.4 or later

import numpy as np
cost = np.zeros(20)
for k in range(2,20):
    kmeans = KMeans()\
            .setK(k)\
            .setSeed(1) \
            .setFeaturesCol("features")\
            .setPredictionCol("cluster")
```

```
    model = kmeans.fit(scaledData)
    cost[k] = model.computeCost(scaledData) # requires Spark 2.0 or later

import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import seaborn as sbs
from matplotlib.ticker import MaxNLocator

fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,20),cost[2:20], marker = "o")
ax.set_xlabel('k')
ax.set_ylabel('cost')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```

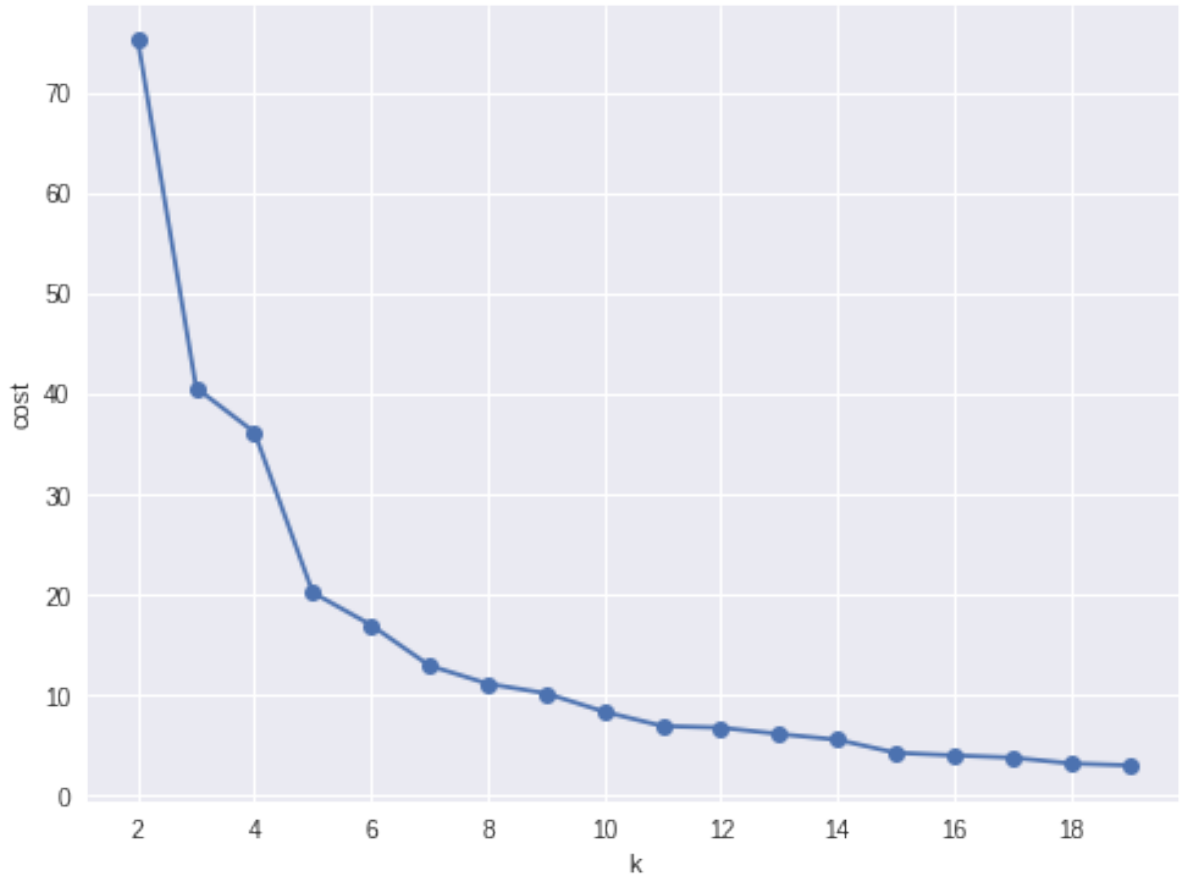


Figure 12.2: Cost v.s. the number of the clusters

In my opinion, sometimes it's hard to choose the number of the clusters. As shown in Figure *Cost v.s. the number of the clusters*, you can choose 3, 5 or even 8. I will choose `3` in this demo.

- Silhouette analysis

```
#PySpark libraries
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import col, percent_rank, lit
```

```
from pyspark.sql.window import Window
from pyspark.sql import DataFrame, Row
from pyspark.sql.types import StructType
from functools import reduce  # For Python 3.x

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

def optimal_k(df_in,index_col,k_min, k_max,num_runs):
    '''
    Determine optimal number of clusters by using Silhoutte Score Analysis.
    :param df_in: the input dataframe
    :param index_col: the name of the index column
    :param k_min: the train dataset
    :param k_min: the minmum number of the clusters
    :param k_max: the maxmum number of the clusters
    :param num_runs: the number of runs for each fixed clusters

    :return k: optimal number of the clusters
    :return silh_lst: Silhouette score
    :return r_table: the running results table

    :author: Wenqiang Feng
    :email:  WFeng@dstsystems.com
    '''

    start = time.time()
    silh_lst = []
    k_lst = np.arange(k_min, k_max+1)

    r_table = df_in.select(index_col).toPandas()
    r_table = r_table.set_index(index_col)
    centers = pd.DataFrame()

    for k in k_lst:
        silh_val = []
        for run in np.arange(1, num_runs+1):

            # Trains a k-means model.
            kmeans = KMeans()\
                    .setK(k)\
                    .setSeed(int(np.random.randint(100, size=1)))
            model = kmeans.fit(df_in)

            # Make predictions
            predictions = model.transform(df_in)
            r_table['cluster_{k}_{run}'.format(k=k, run=run)]= predictions.select('predi

            # Evaluate clustering by computing Silhouette score
            evaluator = ClusteringEvaluator()
            silhouette = evaluator.evaluate(predictions)
            silh_val.append(silhouette)

        silh_array=np.asanyarray(silh_val)
        silh_lst.append(silh_array.mean())

    elapsed =  time.time() - start
```

```
    silhouette = pd.DataFrame(list(zip(k_lst,silh_lst)),columns = ['k', 'silhouette'])

    print('+-------------------------------------------------------------+')
    print("|         The finding optimal k phase took %8.0f s.       |" %(elapsed))
    print('+-------------------------------------------------------------+')


    return k_lst[np.argmax(silh_lst, axis=0)], silhouette , r_table

k, silh_lst, r_table = optimal_k(scaledData,index_col,k_min, k_max,num_runs)
```

```
+-------------------------------------------------------------+
|         The finding optimal k phase took     1783 s.       |
+-------------------------------------------------------------+
```

```
spark.createDataFrame(silh_lst).show()
```

```
+---+------------------+
|  k|        silhouette|
+---+------------------+
|  3|0.8045154385557953|
|  4|0.6993528775512052|
|  5|0.6689286654221447|
|  6|0.6356184024841809|
|  7|0.7174102265711756|
|  8|0.6720861758298997|
|  9| 0.601771359881241|
| 10|0.6292447334578428|
+---+------------------+
```

From the silhouette list, we can choose 3 as the optimal number of the clusters.

Warning: `ClusteringEvaluator` in `pyspark.ml.evaluation` requires Spark 2.4 or later!!

4. K-means clustering

```
k = 3
kmeans = KMeans().setK(k).setSeed(1)
model = kmeans.fit(scaledData)
# Make predictions
predictions = model.transform(scaledData)
predictions.show(5,False)
```

```
+----------+------------------+--------------------+----------+
|CustomerID|               rfm|            features|prediction|
+----------+------------------+--------------------+----------+
|     17420|  [50.0,3.0,598.83]|[0.13404825737265...|         0|
|     16861|  [59.0,3.0,151.65]|[0.15817694369973...|         0|
|     16503|[106.0,5.0,1421.43]|[0.28418230563002...|         2|
|     15727| [16.0,7.0,5178.96]|[0.04289544235924...|         0|
|     17389|[0.0,43.0,31300.08]|[0.0,0.1700404858...|         0|
+----------+------------------+--------------------+----------+
only showing top 5 rows
```

12.3.3 Statistical summary

5. statistical summary

```
results = rfm.join(predictions.select('CustomerID','prediction'),'CustomerID',how='left'
results.show(5)
```

```
+----------+-------+---------+--------+----------+
|CustomerID|Recency|Frequency|Monetary|prediction|
+----------+-------+---------+--------+----------+
|     13098|      1|       41|28658.88|         0|
|     13248|    124|        2|  465.68|         2|
|     13452|    259|        2|   590.0|         1|
|     13460|     29|        2|  183.44|         0|
|     13518|     85|        1|  659.44|         0|
+----------+-------+---------+--------+----------+
only showing top 5 rows
```

- simple summary

```
results.groupBy('prediction')\
       .agg({'Recency':'mean',
             'Frequency': 'mean',
             'Monetary': 'mean'} )\
        .sort(F.col('prediction')).show(5)
```

```
+----------+------------------+------------------+------------------+
|prediction|      avg(Recency)|     avg(Monetary)|    avg(Frequency)|
+----------+------------------+------------------+------------------+
|         0|30.966337980278816|2543.0355321319284| 6.514450867052023|
|         1|296.02403846153845|407.16831730769206|1.5592948717948718|
|         2|154.40148698884758| 702.5096406443623| 2.550185873605948|
+----------+------------------+------------------+------------------+
```

- complex summary

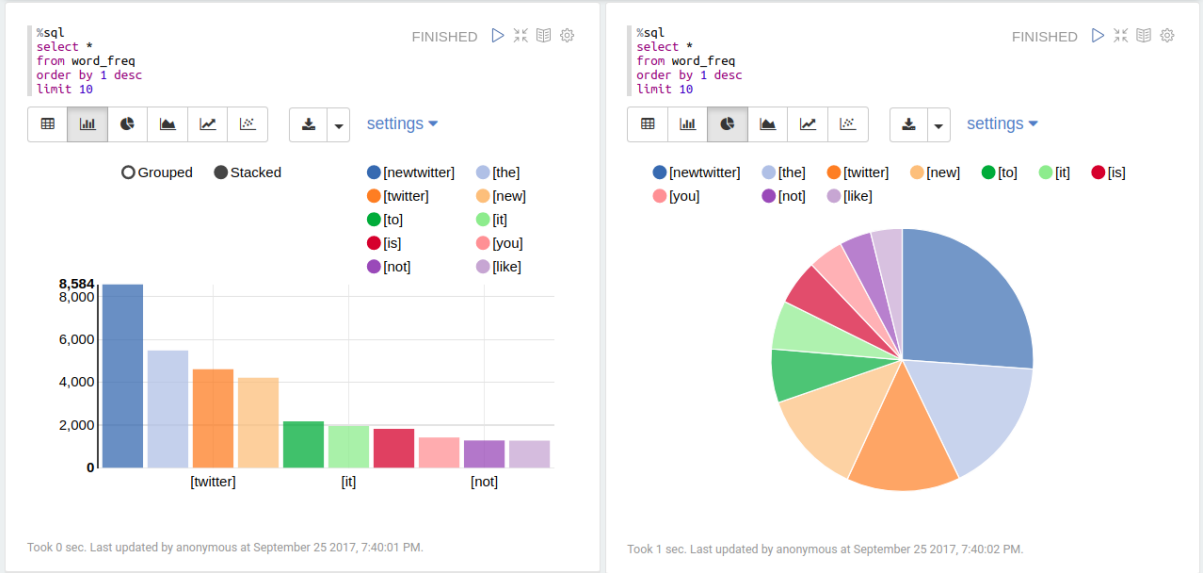
```
grp = 'RFMScore'
num_cols = ['Recency','Frequency','Monetary']
df_input = results

quantile_grouped = quantile_agg(df_input,grp,num_cols)
quantile_grouped.toPandas().to_csv(output_dir+'quantile_grouped.csv')

deciles_grouped = deciles_agg(df_input,grp,num_cols)
deciles_grouped.toPandas().to_csv(output_dir+'deciles_grouped.csv')
```

CHAPTER

THIRTEEN

TEXT MINING

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb



13.1 Text Collection

13.1.1 Image to text

- My `img2txt` function

```
def img2txt(img_dir):
    """
    convert images to text
    """
    import os, PythonMagick
    from datetime import datetime
    import PyPDF2

    from PIL import Image
    import pytesseract

    f = open('doc4img.txt','wa')
    for img in [img_file for img_file in os.listdir(img_dir)
```

```
            if (img_file.endswith(".png") or
                img_file.endswith(".jpg") or
                img_file.endswith(".jpeg"))]:

        start_time = datetime.now()

        input_img = img_dir + "/" + img

        print('----------------------------------------------------------------------')
        print(img)
        print('Converting ' + img +'.......')
        print('----------------------------------------------------------------------')

        # extract the text information from images
        text = pytesseract.image_to_string(Image.open(input_img))
        print(text)

        # ouput text file
        f.write( img + "\n")
        f.write(text.encode('utf-8'))


        print "CPU Time for converting" + img +":"+ str(datetime.now() - start_time) +"\
        f.write( "\n----------------------------------------------------------------\n")

    f.close()
```

- Demo

I applied my `img2txt` function to the image in Image folder.

```
image_dir = r"Image"

img2txt(image_dir)
```

Then I got the following results:

```
----------------------------------------------------------------------
feng.pdf_0.png
Converting feng.pdf_0.png.......
----------------------------------------------------------------------
l I l w

Wenqiang Feng
Data Scientist
DST APPLIED ANALYTICS GROUP



Wenqiang Feng is Data Scientist for DST's Applied Analytics Group. Dr. Feng's responsibi
include providing DST clients with access to cutting--edge skills and technologies, incl
Data analytic solutions, advanced analytic and data enhancement techniques and modeling.

Dr. Feng has deep analytic expertise in data mining, analytic systems, machine learning
algorithms, business intelligence, and applying Big Data tools to strategically solve ir
problems in a cross--functional business. Before joining the DST Applied Analytics Group
Feng holds a MA Data Science Fellow at The Institute for Mathematics and Its Applicatior
(IMA) at the University of Minnesota. While there, he helped startup companies make
```

```
marketing decisions based on deep predictive analytics.

Dr. Feng graduated from University of Tennessee, Knoxville with PhD in Computational
mathematics and Master's degree in Statistics. He also holds Master's degree in Computat
Mathematics at Missouri University of Science and Technology (MST) and Master's degree i
Applied Mathematics at University of science and technology of China (USTC).
CPU Time for convertingfeng.pdf_0.png:0:00:02.061208
```

13.1.2 Image Enhnaced to text

- My `img2txt_enhance` function

```
def img2txt_enhance(img_dir,scaler):
    """
    convert images files to text
    """

    import numpy as np
    import os, PythonMagick
    from datetime import datetime
    import PyPDF2

    from PIL import Image, ImageEnhance, ImageFilter
    import pytesseract

    f = open('doc4img.txt','wa')
    for img in [img_file for img_file in os.listdir(img_dir)
                if (img_file.endswith(".png") or
                    img_file.endswith(".jpg") or
                    img_file.endswith(".jpeg"))]:

        start_time = datetime.now()

        input_img = img_dir + "/" + img
        enhanced_img = img_dir + "/" +"Enhanced" + "/"+ img

        im = Image.open(input_img) # the second one
        im = im.filter(ImageFilter.MedianFilter())
        enhancer = ImageEnhance.Contrast(im)
        im = enhancer.enhance(1)
        im = im.convert('1')
        im.save(enhanced_img)

        for scale in np.ones(scaler):
            im = Image.open(enhanced_img) # the second one
            im = im.filter(ImageFilter.MedianFilter())
            enhancer = ImageEnhance.Contrast(im)
            im = enhancer.enhance(scale)
            im = im.convert('1')
            im.save(enhanced_img)



        print('----------------------------------------------------------------------')
        print(img)
        print('Converting ' + img +'.......')
```

```
        print('------------------------------------------------------------------')

        # extract the text information from images
        text = pytesseract.image_to_string(Image.open(enhanced_img))
        print(text)

        # ouput text file
        f.write( img + "\n")
        f.write(text.encode('utf-8'))


        print "CPU Time for converting" + img +":"+ str(datetime.now() - start_time) +"\
        f.write( "\n------------------------------------------------------------------\n")

    f.close()
```

- Demo

I applied my `img2txt_enhance` function to the following noised image in Enhance folder.

```
image_dir = r"Enhance"

pdf2txt_enhance(image_dir)
```

Then I got the following results:

```
------------------------------------------------------------------
noised.jpg
Converting noised.jpg.......
------------------------------------------------------------------
zHHH
CPU Time for convertingnoised.jpg:0:00:00.135465
```

while the result from `img2txt` function is

```
------------------------------------------------------------------
noised.jpg
Converting noised.jpg.......
------------------------------------------------------------------
,2 WW
CPU Time for convertingnoised.jpg:0:00:00.133508
```

which is not correct.

13.1.3 PDF to text

- My `pdf2txt` function

```
def pdf2txt(pdf_dir,image_dir):
    """
    convert PDF to text
```

```
    """

    import os, PythonMagick
    from datetime import datetime
    import PyPDF2

    from PIL import Image
    import pytesseract

    f = open('doc.txt','wa')
    for pdf in [pdf_file for pdf_file in os.listdir(pdf_dir) if pdf_file.endswith(".pdf"

        start_time = datetime.now()

        input_pdf = pdf_dir + "/" + pdf

        pdf_im = PyPDF2.PdfFileReader(file(input_pdf, "rb"))
        npage = pdf_im.getNumPages()

        print('------------------------------------------------------------------------')
        print(pdf)
        print('Converting %d pages.' % npage)
        print('------------------------------------------------------------------------')

        f.write( "\n------------------------------------------------------------------------

        for p in range(npage):

            pdf_file = input_pdf + '[' + str(p) +']'
            image_file =  image_dir  + "/" + pdf+ '_' + str(p)+ '.png'

            # convert PDF files to Images
            im = PythonMagick.Image()
            im.density('300')
            im.read(pdf_file)
            im.write(image_file)

            # extract the text information from images
            text = pytesseract.image_to_string(Image.open(image_file))

            #print(text)

            # ouput text file
            f.write( pdf + "\n")
            f.write(text.encode('utf-8'))


        print "CPU Time for converting" + pdf +":"+ str(datetime.now() - start_time) +"\

    f.close()
```

- Demo

I applied my `pdf2txt` function to my scaned bio pdf file in pdf folder.

```
pdf_dir = r"pdf"
image_dir = r"Image"
```

```
pdf2txt(pdf_dir,image_dir)
```

Then I got the following results:

```
--------------------------------------------------------------------------
feng.pdf
Converting 1 pages.
--------------------------------------------------------------------------
l I l w

Wenqiang Feng
Data Scientist
DST APPLIED ANALYTICS GROUP



Wenqiang Feng is Data Scientist for DST's Applied Analytics Group. Dr. Feng's responsibi
include providing DST clients with access to cutting--edge skills and technologies, incl
Data analytic solutions, advanced analytic and data enhancement techniques and modeling.

Dr. Feng has deep analytic expertise in data mining, analytic systems, machine learning
algorithms, business intelligence, and applying Big Data tools to strategically solve ir
problems in a cross--functional business. Before joining the DST Applied Analytics Group
Feng holds a MA Data Science Fellow at The Institute for Mathematics and Its Applicatior
(IMA) at the University of Minnesota. While there, he helped startup companies make
marketing decisions based on deep predictive analytics.

Dr. Feng graduated from University of Tennessee, Knoxville with PhD in Computational
mathematics and Master's degree in Statistics. He also holds Master's degree in Computat
Mathematics at Missouri University of Science and Technology (MST) and Master's degree i
Applied Mathematics at University of science and technology of China (USTC).
CPU Time for convertingfeng.pdf:0:00:03.143800
```

13.1.4 Audio to text

- My `audio2txt` function

```
def audio2txt(audio_dir):
    ''' convert audio to text'''

    import speech_recognition as sr
    r = sr.Recognizer()

    f = open('doc.txt','wa')
    for audio_n in [audio_file for audio_file in os.listdir(audio_dir) \
                    if audio_file.endswith(".wav")]:

        filename = audio_dir + "/" + audio_n

        # Read audio data
        with sr.AudioFile(filename) as source:
            audio = r.record(source)  # read the entire audio file

        # Google Speech Recognition
        text = r.recognize_google(audio)

        # ouput text file
```

```
            f.write( audio_n + ": ")
            f.write(text.encode('utf-8'))
            f.write("\n")

            print('You said: ' + text)

    f.close()
```

- Demo

I applied my `audio2txt` function to my audio records in audio folder.

```
audio_dir = r"audio"

audio2txt(audio_dir)
```

Then I got the following results:

```
You said: hello this is George welcome to my tutorial
You said: mathematics is important in daily life
You said: call me tomorrow
You said: do you want something to eat
You said: I want to speak with him
You said: nice to see you
You said: can you speak slowly
You said: have a good day
```

By the way, you can use my following python code to record your own audio and play with `audio2txt` function in Command-line `python record.py "demo2.wav"`:

```
import sys, getopt

import speech_recognition as sr

audio_filename = sys.argv[1]

r = sr.Recognizer()
with sr.Microphone() as source:
    r.adjust_for_ambient_noise(source)
    print("Hey there, say something, I am recording!")
    audio = r.listen(source)
    print("Done listening!")

with open(audio_filename, "wb") as f:
    f.write(audio.get_wav_data())
```

13.2 Text Preprocessing

- check to see if a row only contains whitespace

```
def check_blanks(data_str):
    is_blank = str(data_str.isspace())
    return is_blank
```

- Determine whether the language of the text content is english or not: Use langid module to classify the language to make sure we are applying the correct cleanup actions for English langid

```
def check_lang(data_str):
    predict_lang = langid.classify(data_str)
    if predict_lang[1] >= .9:
        language = predict_lang[0]
    else:
        language = 'NA'
    return language
```

- Remove features

```
def remove_features(data_str):
    # compile regex
    url_re = re.compile('https?://(www.)?\w+\.\w+(/\w+)*/?')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\\d+)')
    mention_re = re.compile('@(\w+)')
    alpha_num_re = re.compile("^[a-z0-9_.]+$")
    # convert to lowercase
    data_str = data_str.lower()
    # remove hyperlinks
    data_str = url_re.sub(' ', data_str)
    # remove @mentions
    data_str = mention_re.sub(' ', data_str)
    # remove puncuation
    data_str = punc_re.sub(' ', data_str)
    # remove numeric 'words'
    data_str = num_re.sub(' ', data_str)
    # remove non a-z 0-9 characters and words shorter than 3 characters
    list_pos = 0
    cleaned_str = ''
    for word in data_str.split():
        if list_pos == 0:
            if alpha_num_re.match(word) and len(word) > 2:
                cleaned_str = word
            else:
                cleaned_str = ' '
        else:
            if alpha_num_re.match(word) and len(word) > 2:
                cleaned_str = cleaned_str + ' ' + word
            else:
                cleaned_str += ' '
        list_pos += 1
    return cleaned_str
```

- removes stop words

```
def remove_stops(data_str):
    # expects a string
    stops = set(stopwords.words("english"))
    list_pos = 0
    cleaned_str = ''
    text = data_str.split()
    for word in text:
        if word not in stops:
            # rebuild cleaned_str
            if list_pos == 0:
                cleaned_str = word
            else:
```

```
                cleaned_str = cleaned_str + ' ' + word
            list_pos += 1
    return cleaned_str
```

- tagging text

```
def tag_and_remove(data_str):
    cleaned_str = ' '
    # noun tags
    nn_tags = ['NN', 'NNP', 'NNP', 'NNPS', 'NNS']
    # adjectives
    jj_tags = ['JJ', 'JJR', 'JJS']
    # verbs
    vb_tags = ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']
    nltk_tags = nn_tags + jj_tags + vb_tags

    # break string into 'words'
    text = data_str.split()

    # tag the text and keep only those with the right tags
    tagged_text = pos_tag(text)
    for tagged_word in tagged_text:
        if tagged_word[1] in nltk_tags:
            cleaned_str += tagged_word[0] + ' '

    return cleaned_str
```

- lemmatization

```
def lemmatize(data_str):
    # expects a string
    list_pos = 0
    cleaned_str = ''
    lmtzr = WordNetLemmatizer()
    text = data_str.split()
    tagged_words = pos_tag(text)
    for word in tagged_words:
        if 'v' in word[1].lower():
            lemma = lmtzr.lemmatize(word[0], pos='v')
        else:
            lemma = lmtzr.lemmatize(word[0], pos='n')
        if list_pos == 0:
            cleaned_str = lemma
        else:
            cleaned_str = cleaned_str + ' ' + lemma
        list_pos += 1
    return cleaned_str
```

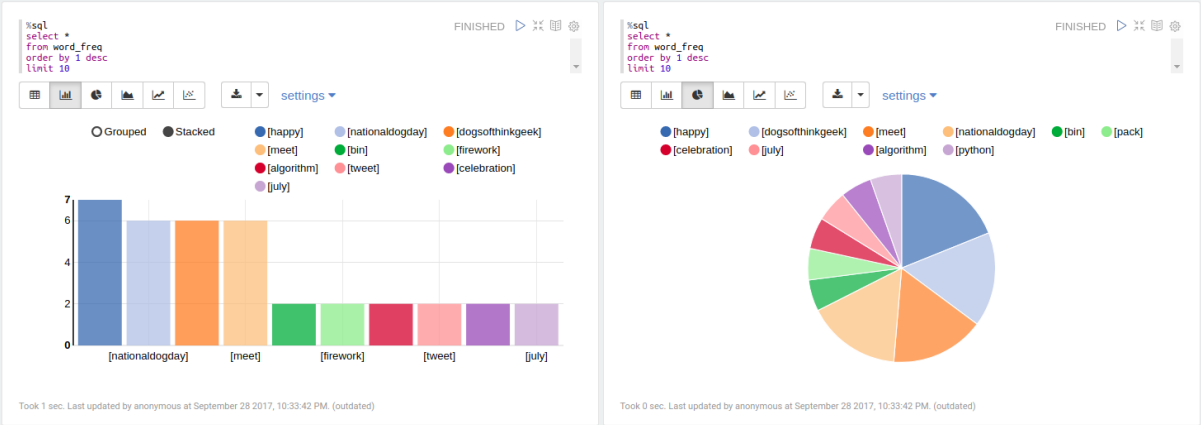
define the preprocessing function in PySpark

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import preproc as pp

check_lang_udf = udf(pp.check_lang, StringType())
remove_stops_udf = udf(pp.remove_stops, StringType())
remove_features_udf = udf(pp.remove_features, StringType())
tag_and_remove_udf = udf(pp.tag_and_remove, StringType())
```

```
lemmatize_udf = udf(pp.lemmatize, StringType())
check_blanks_udf = udf(pp.check_blanks, StringType())
```

13.3 Text Classification

Theoretically speaking, you may apply any classification algorithms to do classification. I will only present Naive Bayes method is the following.



13.3.1 Introduction

13.3.2 Demo

1. create spark contexts

```
import pyspark
from pyspark.sql import SQLContext

# create spark contexts
sc = pyspark.SparkContext()
sqlContext = SQLContext(sc)
```

2. load dataset

```
# Load a text file and convert each line to a Row.
data_rdd = sc.textFile("../data/raw_data.txt")
parts_rdd = data_rdd.map(lambda l: l.split("\t"))

# Filter bad rows out
garantee_col_rdd = parts_rdd.filter(lambda l: len(l) == 3)
typed_rdd = garantee_col_rdd.map(lambda p: (p[0], p[1], float(p[2])))

#Create DataFrame
data_df = sqlContext.createDataFrame(typed_rdd, ["text", "id", "label"])

# get the raw columns
raw_cols = data_df.columns

#data_df.show()
data_df.printSchema()
```

```
root
 |-- text: string (nullable = true)
 |-- id: string (nullable = true)
 |-- label: double (nullable = true)

+--------------------+------------------+-----+
|                text|                id|label|
+--------------------+------------------+-----+
|Fresh install of ...|        1018769417|  1.0|
|Well. Now I know ...|       10284216536|  1.0|
|"Literally six we...|       10298589026|  1.0|
|Mitsubishi i MiEV...|109017669432377344|  1.0|
+--------------------+------------------+-----+
only showing top 4 rows
```

3. setup pyspark udf function

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import preproc as pp

# Register all the functions in Preproc with Spark Context
check_lang_udf = udf(pp.check_lang, StringType())
remove_stops_udf = udf(pp.remove_stops, StringType())
remove_features_udf = udf(pp.remove_features, StringType())
tag_and_remove_udf = udf(pp.tag_and_remove, StringType())
lemmatize_udf = udf(pp.lemmatize, StringType())
check_blanks_udf = udf(pp.check_blanks, StringType())
```

4. language identification

```
lang_df = data_df.withColumn("lang", check_lang_udf(data_df["text"]))
en_df = lang_df.filter(lang_df["lang"] == "en")
en_df.show(4)

+--------------------+------------------+-----+----+
|                text|                id|label|lang|
+--------------------+------------------+-----+----+
|RT @goeentertain:...|665305154954989568|  1.0|  en|
|Teforia Uses Mach...|660668007975268352|  1.0|  en|
|   Apple TV or Roku?|       25842461136|  1.0|  en|
|Finished http://t...|        9412369614|  1.0|  en|
+--------------------+------------------+-----+----+
only showing top 4 rows
```

5. remove stop words

```
rm_stops_df = en_df.select(raw_cols)\
                   .withColumn("stop_text", remove_stops_udf(en_df["text"]))
rm_stops_df.show(4)

+--------------------+------------------+-----+--------------------+
|                text|                id|label|           stop_text|
+--------------------+------------------+-----+--------------------+
|RT @goeentertain:...|665305154954989568|  1.0|RT @goeentertain:...|
|Teforia Uses Mach...|660668007975268352|  1.0|Teforia Uses Mach...|
|   Apple TV or Roku?|       25842461136|  1.0|      Apple TV Roku?|
|Finished http://t...|        9412369614|  1.0|Finished http://t...|
```

```
+--------------------+------------------+-----+--------------------+
only showing top 4 rows
```

6. remove irrelevant features

```
rm_features_df = rm_stops_df.select(raw_cols+["stop_text"])\
                            .withColumn("feat_text", \
                            remove_features_udf(rm_stops_df["stop_text"]))
rm_features_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+--------------------
|                text|                id|label|           stop_text|           feat_text
+--------------------+------------------+-----+--------------------+--------------------
|RT @goeentertain:...|665305154954989568|  1.0|RT @goeentertain:...|  future blase   ...
|Teforia Uses Mach...|660668007975268352|  1.0|Teforia Uses Mach...|teforia uses mach...
|   Apple TV or Roku?|       25842461136|  1.0|      Apple TV Roku?|         apple  roku
|Finished http://t...|        9412369614|  1.0|Finished http://t...|            finished
+--------------------+------------------+-----+--------------------+--------------------
only showing top 4 rows
```

7. tag the words

```
tagged_df = rm_features_df.select(raw_cols+["feat_text"]) \
                          .withColumn("tagged_text", \
                           tag_and_remove_udf(rm_features_df.feat_text))

tagged_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+--------------------
|                text|                id|label|           feat_text|         tagged_text
+--------------------+------------------+-----+--------------------+--------------------
|RT @goeentertain:...|665305154954989568|  1.0|  future blase   ...| future blase vic...
|Teforia Uses Mach...|660668007975268352|  1.0|teforia uses mach...| teforia uses mac...
|   Apple TV or Roku?|       25842461136|  1.0|         apple  roku|         apple roku
|Finished http://t...|        9412369614|  1.0|            finished|           finished
+--------------------+------------------+-----+--------------------+--------------------
only showing top 4 rows
```

8. lemmatization of words

```
lemm_df = tagged_df.select(raw_cols+["tagged_text"]) \
                   .withColumn("lemm_text", lemmatize_udf(tagged_df["tagged_text"]))
lemm_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+--------------------
|                text|                id|label|         tagged_text|           lemm_text
+--------------------+------------------+-----+--------------------+--------------------
|RT @goeentertain:...|665305154954989568|  1.0| future blase vic...|future blase vice...
|Teforia Uses Mach...|660668007975268352|  1.0| teforia uses mac...|teforia use machi...
|   Apple TV or Roku?|       25842461136|  1.0|         apple roku |          apple roku
|Finished http://t...|        9412369614|  1.0|           finished |              finish
+--------------------+------------------+-----+--------------------+--------------------
only showing top 4 rows
```

9. remove blank rows and drop duplicates

```
check_blanks_df = lemm_df.select(raw_cols+["lemm_text"])\
                         .withColumn("is_blank", check_blanks_udf(lemm_df["lemm_text
# remove blanks
```

```
    no_blanks_df = check_blanks_df.filter(check_blanks_df["is_blank"] == "False")

# drop duplicates
dedup_df = no_blanks_df.dropDuplicates(['text', 'label'])

    dedup_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+--------+
|                text|                id|label|           lemm_text|is_blank|
+--------------------+------------------+-----+--------------------+--------+
|RT @goeentertain:...|665305154954989568|  1.0|future blase vice...|   False|
|Teforia Uses Mach...|660668007975268352|  1.0|teforia use machi...|   False|
|   Apple TV or Roku?|       25842461136|  1.0|          apple roku|   False|
|Finished http://t...|        9412369614|  1.0|              finish|   False|
+--------------------+------------------+-----+--------------------+--------+
only showing top 4 rows
```

10. add unieuq ID

```
from pyspark.sql.functions import monotonically_increasing_id
# Create Unique ID
dedup_df = dedup_df.withColumn("uid", monotonically_increasing_id())
dedup_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+--------+----------
|                text|                id|label|           lemm_text|is_blank|        ui
+--------------------+------------------+-----+--------------------+--------+----------
|              dragon|        1546813742|  1.0|              dragon|   False| 8589934592
|           hurt much|        1558492525|  1.0|           hurt much|   False|1116691496
|seth blog word se...|383221484023709697|  1.0|seth blog word se...|   False|1288490188
|teforia use machi...|660668007975268352|  1.0|teforia use machi...|   False|1374389534
+--------------------+------------------+-----+--------------------+--------+----------
only showing top 4 rows
```

11. create final dataset

```
data = dedup_df.select('uid','id', 'text','label')
data.show(4)
```

```
+-----------+------------------+--------------------+-----+
|        uid|                id|                text|label|
+-----------+------------------+--------------------+-----+
| 85899345920|        1546813742|              dragon|  1.0|
|111669149696|        1558492525|           hurt much|  1.0|
|128849018880|383221484023709697|seth blog word se...|  1.0|
|137438953472|660668007975268352|teforia use machi...|  1.0|
+-----------+------------------+--------------------+-----+
only showing top 4 rows
```

12. Create taining and test sets

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = data.randomSplit([0.6, 0.4])
```

13. NaiveBayes Pipeline

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml import Pipeline
```

```
from pyspark.ml.classification import NaiveBayes, RandomForestClassifier
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.feature import CountVectorizer

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and nb.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="rawFeatures")
# vectorizer = CountVectorizer(inputCol= "words", outputCol="rawFeatures")
idf = IDF(minDocFreq=3, inputCol="rawFeatures", outputCol="features")

# Naive Bayes model
nb = NaiveBayes()

# Pipeline Architecture
pipeline = Pipeline(stages=[tokenizer, hashingTF, idf, nb])

# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

14. Make predictions

```
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("text", "label", "prediction").show(5,False)
```

```
+-----------------------------------------------+-----+----------+
|text                                           |label|prediction|
+-----------------------------------------------+-----+----------+
|finish                                         |1.0  |1.0       |
|meet rolo dogsofthinkgeek happy nationaldogday |1.0  |1.0       |
|pumpkin family                                 |1.0  |1.0       |
|meet jet dogsofthinkgeek happy nationaldogday  |1.0  |1.0       |
|meet vixie dogsofthinkgeek happy nationaldogday|1.0  |1.0       |
+-----------------------------------------------+-----+----------+
only showing top 5 rows
```

15. evaluation

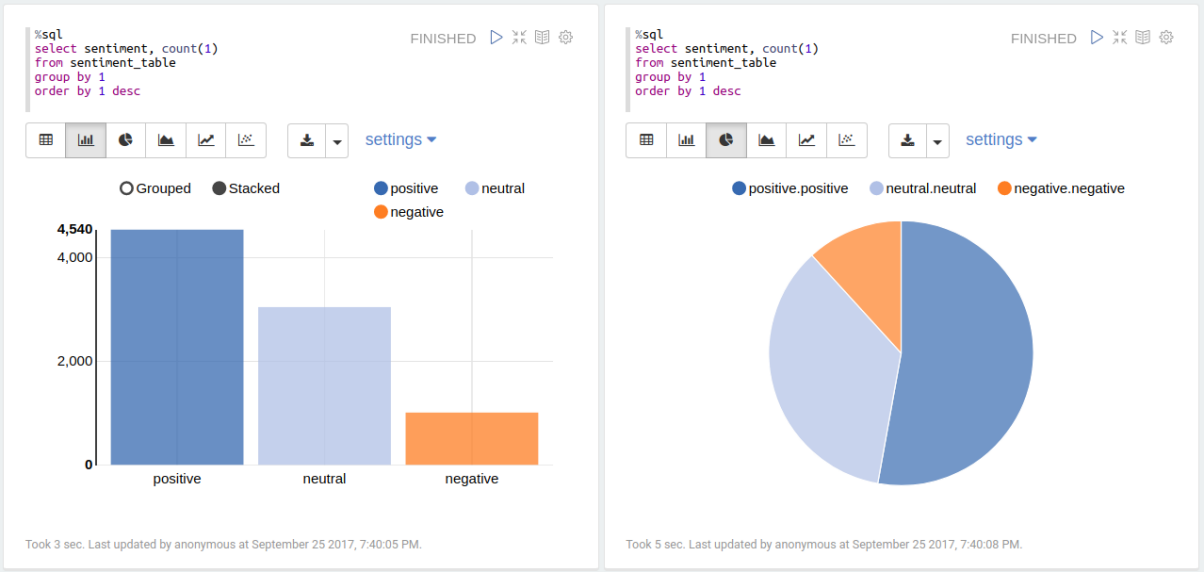
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
evaluator.evaluate(predictions)
```

```
0.912655971479501
```

13.4 Sentiment analysis

13.4.1 Introduction

Sentiment analysis (sometimes known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify,



extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine.

Generally speaking, sentiment analysis aims to **determine the attitude** of a speaker, writer, or other subject with respect to some topic or the overall contextual polarity or emotional reaction to a document, interaction, or event. The attitude may be a judgment or evaluation (see appraisal theory), affective state (that is to say, the emotional state of the author or speaker), or the intended emotional communication (that is to say, the emotional effect intended by the author or interlocutor).

Sentiment analysis in business, also known as opinion mining is a process of identifying and cataloging a piece of text according to the tone conveyed by it. It has broad application:

- Sentiment Analysis in Business Intelligence Build up
- Sentiment Analysis in Business for Competitive Advantage
- Enhancing the Customer Experience through Sentiment Analysis in Business

13.4.2 Pipeline

Figure 13.1: Sentiment Analysis Pipeline

13.4.3 Demo

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark Sentiment Analysis example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                                options(header='true', \
                                inferschema='true').\
            load("../data/newtwitter.csv",header=True);
```

```
+--------------------+----------+-------+
|                text|        id|pubdate|
+--------------------+----------+-------+
|10 Things Missing...|2602860537|  18536|
|RT @_NATURALBWINN...|2602850443|  18536|
|RT @HBO24 yo the ...|2602761852|  18535|
|Aaaaaaaand I have...|2602738438|  18535|
|can I please have...|2602684185|  18535|
+--------------------+----------+-------+
only showing top 5 rows
```

3. Text Preprocessing

- remove non ASCII characters

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import pos_tag
import string
import re

# remove non ASCII characters
def strip_non_ascii(data_str):
    ''' Returns the string without non ASCII characters'''
    stripped = (c for c in data_str if 0 < ord(c) < 127)
    return ''.join(stripped)
# setup pyspark udf function
strip_non_ascii_udf = udf(strip_non_ascii, StringType())
```

check:

```
df = df.withColumn('text_non_asci',strip_non_ascii_udf(df['text']))
df.show(5,True)
```

ouput:

```
+--------------------+----------+-------+--------------------+
|                text|        id|pubdate|       text_non_asci|
+--------------------+----------+-------+--------------------+
|10 Things Missing...|2602860537|  18536|10 Things Missing...|
|RT @_NATURALBWINN...|2602850443|  18536|RT @_NATURALBWINN...|
```

```
|RT @HBO24 yo the ...|2602761852|  18535|RT @HBO24 yo the ...|
|Aaaaaaaand I have...|2602738438|  18535|Aaaaaaaand I have...|
|can I please have...|2602684185|  18535|can I please have...|
+--------------------+----------+-------+--------------------+
only showing top 5 rows
```

- fixed abbreviation

```
# fixed abbreviation
def fix_abbreviation(data_str):
    data_str = data_str.lower()
    data_str = re.sub(r'\bthats\b', 'that is', data_str)
    data_str = re.sub(r'\bive\b', 'i have', data_str)
    data_str = re.sub(r'\bim\b', 'i am', data_str)
    data_str = re.sub(r'\bya\b', 'yeah', data_str)
    data_str = re.sub(r'\bcant\b', 'can not', data_str)
    data_str = re.sub(r'\bdont\b', 'do not', data_str)
    data_str = re.sub(r'\bwont\b', 'will not', data_str)
    data_str = re.sub(r'\bid\b', 'i would', data_str)
    data_str = re.sub(r'wtf', 'what the fuck', data_str)
    data_str = re.sub(r'\bwth\b', 'what the hell', data_str)
    data_str = re.sub(r'\br\b', 'are', data_str)
    data_str = re.sub(r'\bu\b', 'you', data_str)
    data_str = re.sub(r'\bk\b', 'OK', data_str)
    data_str = re.sub(r'\bsux\b', 'sucks', data_str)
    data_str = re.sub(r'\bno+\b', 'no', data_str)
    data_str = re.sub(r'\bcoo+\b', 'cool', data_str)
    data_str = re.sub(r'rt\b', '', data_str)
    data_str = data_str.strip()
    return data_str

fix_abbreviation_udf = udf(fix_abbreviation, StringType())
```

check:

```
    df = df.withColumn('fixed_abbrev',fix_abbreviation_udf(df['text_non_asci']))
    df.show(5,True)
```

ouput:

```
+--------------------+----------+-------+--------------------+--------------------+
|                text|        id|pubdate|       text_non_asci|        fixed_abbrev|
+--------------------+----------+-------+--------------------+--------------------+
|10 Things Missing...|2602860537|  18536|10 Things Missing...|10 things missing...|
|RT @_NATURALBWINN...|2602850443|  18536|RT @_NATURALBWINN...|@_naturalbwinner ...|
|RT @HBO24 yo the ...|2602761852|  18535|RT @HBO24 yo the ...|@hbo24 yo the #ne.../
|Aaaaaaaand I have...|2602738438|  18535|Aaaaaaaand I have...|aaaaaaaand i have...|
|can I please have...|2602684185|  18535|can I please have...|can i please have...|
+--------------------+----------+-------+--------------------+--------------------+
only showing top 5 rows
```

- remove irrelevant features

```
def remove_features(data_str):
    # compile regex
    url_re = re.compile('https?://(www.)?\w+\.\w+(/\w+)*/?')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\\d+)')
```

```
    mention_re = re.compile('@(\w+)')
    alpha_num_re = re.compile("^[a-z0-9_.]+$")
    # convert to lowercase
    data_str = data_str.lower()
    # remove hyperlinks
    data_str = url_re.sub(' ', data_str)
    # remove @mentions
    data_str = mention_re.sub(' ', data_str)
    # remove puncuation
    data_str = punc_re.sub(' ', data_str)
    # remove numeric 'words'
    data_str = num_re.sub(' ', data_str)
    # remove non a-z 0-9 characters and words shorter than 1 characters
    list_pos = 0
    cleaned_str = ''
    for word in data_str.split():
        if list_pos == 0:
            if alpha_num_re.match(word) and len(word) > 1:
                cleaned_str = word
            else:
                cleaned_str = ' '
        else:
            if alpha_num_re.match(word) and len(word) > 1:
                cleaned_str = cleaned_str + ' ' + word
            else:
                cleaned_str += ' '
        list_pos += 1
    # remove unwanted space, *.split() will automatically split on
    # whitespace and discard duplicates, the " ".join() joins the
    # resulting list into one string.
    return " ".join(cleaned_str.split())
# setup pyspark udf function
remove_features_udf = udf(remove_features, StringType())
```

check:

```
df = df.withColumn('removed',remove_features_udf(df['fixed_abbrev']))
df.show(5,True)
```

ouput:

```
+--------------------+----------+-------+--------------------+--------------------+-----
|                text|        id|pubdate|       text_non_asci|        fixed_abbrev|
+--------------------+----------+-------+--------------------+--------------------+-----
|10 Things Missing...|2602860537|  18536|10 Things Missing...|10 things missing...|thing
|RT @_NATURALBWINN...|2602850443|  18536|RT @_NATURALBWINN...|@_naturalbwinner ...|oh ar
|RT @HBO24 yo the ...|2602761852|  18535|RT @HBO24 yo the ...|@hbo24 yo the #ne...|yo th
|Aaaaaaaand I have...|2602738438|  18535|Aaaaaaaand I have...|aaaaaaaand i have...|aaaaa
|can I please have...|2602684185|  18535|can I please have...|can i please have...|can p
+--------------------+----------+-------+--------------------+--------------------+-----
only showing top 5 rows
```

4. Sentiment Analysis main function

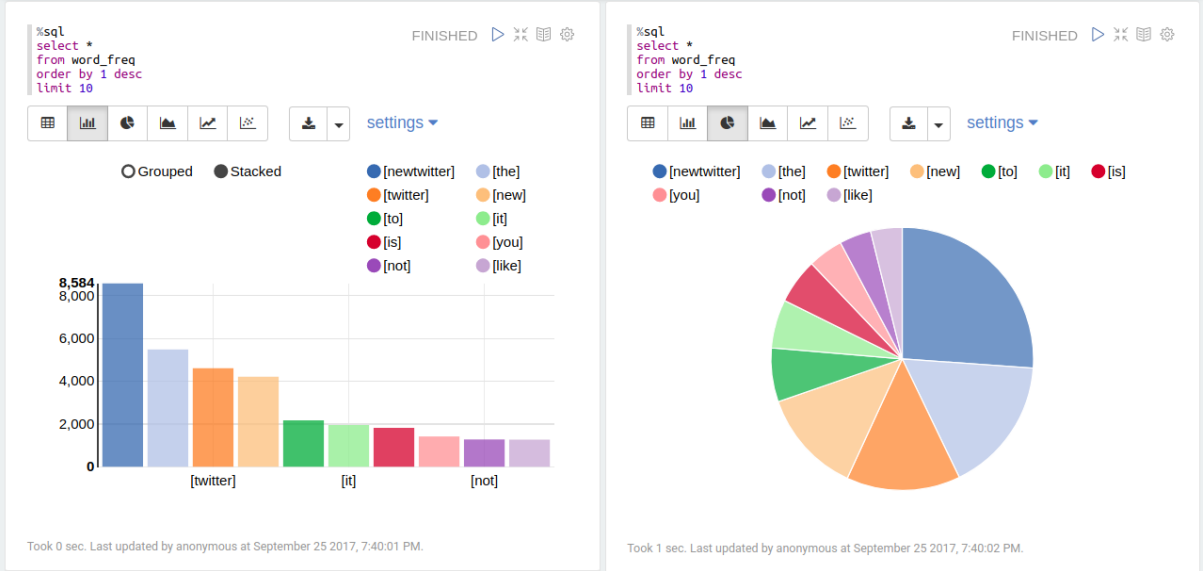
```
from pyspark.sql.types import FloatType

from textblob import TextBlob
```

```
def sentiment_analysis(text):
    return TextBlob(text).sentiment.polarity

sentiment_analysis_udf = udf(sentiment_analysis , FloatType())

df  = df.withColumn("sentiment_score", sentiment_analysis_udf( df['removed'] ))
df.show(5,True)
```

- Sentiment score

```
+--------------------+---------------+
|             removed|sentiment_score|
+--------------------+---------------+
|things missing in...|    -0.03181818|
|oh and do not lik...|    -0.03181818|
|yo the newtwitter...|     0.3181818|
|aaaaaaaand have t...|     0.11818182|
|can please have t...|     0.13636364|
+--------------------+---------------+
only showing top 5 rows
```

- Words frequency

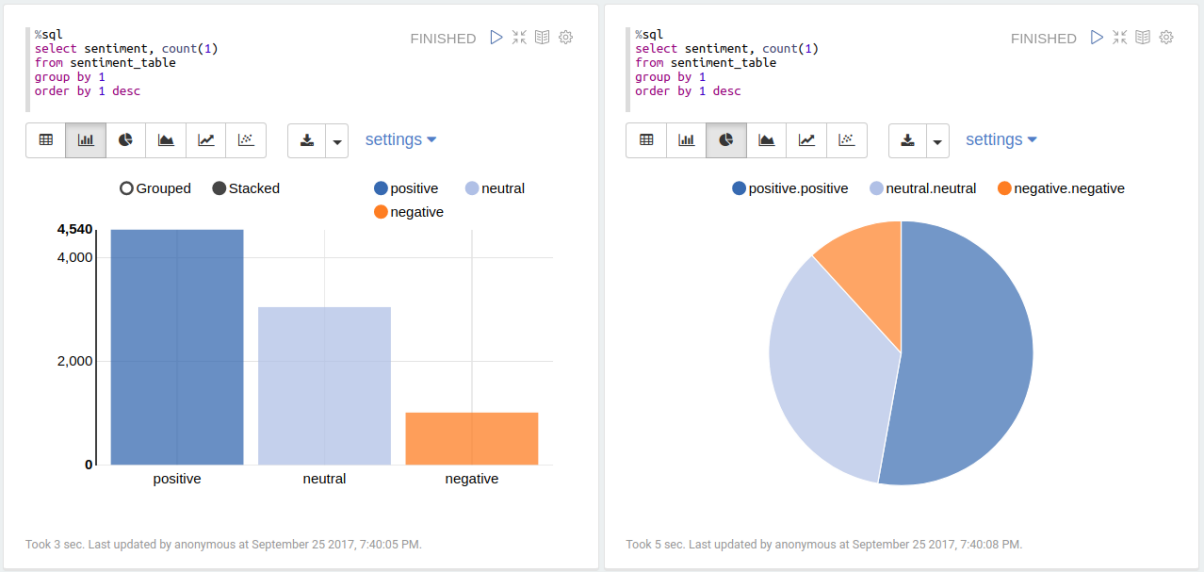


- Sentiment Classification

```
def condition(r):
    if (r >=0.1):
        label = "positive"
    elif(r <= -0.1):
        label = "negative"
    else:
        label = "neutral"
    return label

sentiment_udf = udf(lambda x: condition(x), StringType())
```

5. Output

- Sentiment Class

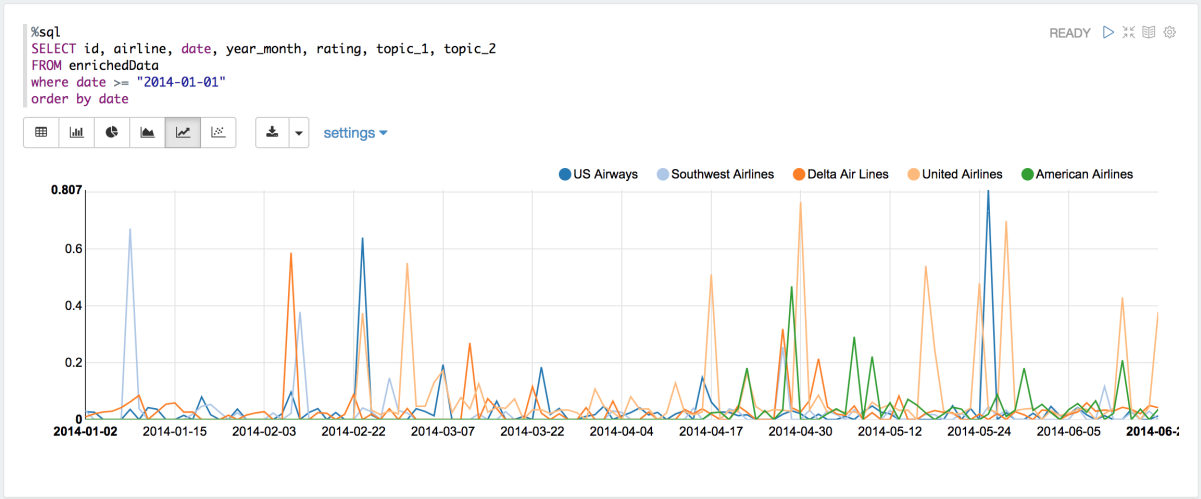


- Top tweets from each sentiment class

```
+--------------------+---------------+---------+
|                text|sentiment_score|sentiment|
+--------------------+---------------+---------+
|and this #newtwit...|            1.0| positive|
|"RT @SarahsJokes:...|            1.0| positive|
|#newtwitter using...|            1.0| positive|
|The #NewTwitter h...|            1.0| positive|
|You can now undo ...|            1.0| positive|
+--------------------+---------------+---------+
only showing top 5 rows

+--------------------+---------------+---------+
|                text|sentiment_score|sentiment|
+--------------------+---------------+---------+
|Lists on #NewTwit...|           -0.1|  neutral|
|Too bad most of m...|           -0.1|  neutral|
|the #newtwitter i...|           -0.1|  neutral|
|Looks like our re...|           -0.1|  neutral|
|i switched to the...|           -0.1|  neutral|
+--------------------+---------------+---------+
only showing top 5 rows

+--------------------+---------------+---------+
|                text|sentiment_score|sentiment|
+--------------------+---------------+---------+
|oh. #newtwitter i...|           -1.0| negative|
|RT @chqwn: #NewTw...|           -1.0| negative|
|Copy that - its W...|           -1.0| negative|
|RT @chqwn: #NewTw...|           -1.0| negative|
|#NewTwitter has t...|           -1.0| negative|
+--------------------+---------------+---------+
only showing top 5 rows
```

13.5 N-grams and Correlations

13.6 Topic Model: Latent Dirichlet Allocation



13.6.1 Introduction

In text mining, a topic model is a unsupervised model for discovering the abstract "topics" that occur in a collection of documents.

Latent Dirichlet Allocation (LDA) is a mathematical method for estimating both of these at the same time: finding the mixture of words that is associated with each topic, while also determining the mixture of topics that describes each document.

13.6.2 Demo

1. Load data

```
rawdata = spark.read.load("../data/airlines.csv", format="csv", header=True)
rawdata.show(5)
```

```
+-----+---------------+---------+--------+------+--------+-----+-----------+-------
|   id|        airline|     date|location|rating|   cabin|value|recommended|
+-----+---------------+---------+--------+------+--------+-----+-----------+-------
|10001|Delta Air Lines|21-Jun-14|Thailand|     7| Economy|    4|        YES|Flew Ma
|10002|Delta Air Lines|19-Jun-14|     USA|     0| Economy|    2|         NO|Flight
|10003|Delta Air Lines|18-Jun-14|     USA|     0| Economy|    1|         NO|Delta W
|10004|Delta Air Lines|17-Jun-14|     USA|     9|Business|    4|        YES|"I just
|10005|Delta Air Lines|17-Jun-14| Ecuador|     7| Economy|    3|        YES|"Round-
+-----+---------------+---------+--------+------+--------+-----+-----------+-------
only showing top 5 rows
```

1. Text preprocessing

I will use the following raw column names to keep my table concise:

```
raw_cols =  rawdata.columns
raw_cols

['id', 'airline', 'date', 'location', 'rating', 'cabin', 'value', 'recommended', 'r

rawdata = rawdata.dropDuplicates(['review'])

from pyspark.sql.functions import udf, col
from pyspark.sql.types import StringType, DoubleType, DateType

from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import pos_tag
import langid
import string
import re
```

- remove non ASCII characters

```
# remove non ASCII characters
def strip_non_ascii(data_str):
    ''' Returns the string without non ASCII characters'''
    stripped = (c for c in data_str if 0 < ord(c) < 127)
    return ''.join(stripped)
```

- check it blank line or not

```
# check to see if a row only contains whitespace
def check_blanks(data_str):
    is_blank = str(data_str.isspace())
    return is_blank
```

- check the language (a little bit slow, I skited this step)

```
# check the language (only apply to english)
def check_lang(data_str):
    from langid.langid import LanguageIdentifier, model
    identifier = LanguageIdentifier.from_modelstring(model, norm_probs=True)
    predict_lang = identifier.classify(data_str)

    if predict_lang[1] >= .9:
        language = predict_lang[0]
    else:
        language = predict_lang[0]
    return language
```

- fixed abbreviation

```
# fixed abbreviation
def fix_abbreviation(data_str):
    data_str = data_str.lower()
    data_str = re.sub(r'\bthats\b', 'that is', data_str)
    data_str = re.sub(r'\bive\b', 'i have', data_str)
    data_str = re.sub(r'\bim\b', 'i am', data_str)
    data_str = re.sub(r'\bya\b', 'yeah', data_str)
    data_str = re.sub(r'\bcant\b', 'can not', data_str)
    data_str = re.sub(r'\bdont\b', 'do not', data_str)
    data_str = re.sub(r'\bwont\b', 'will not', data_str)
```

```
    data_str = re.sub(r'\bid\b', 'i would', data_str)
    data_str = re.sub(r'wtf', 'what the fuck', data_str)
    data_str = re.sub(r'\bwth\b', 'what the hell', data_str)
    data_str = re.sub(r'\br\b', 'are', data_str)
    data_str = re.sub(r'\bu\b', 'you', data_str)
    data_str = re.sub(r'\bk\b', 'OK', data_str)
    data_str = re.sub(r'\bsux\b', 'sucks', data_str)
    data_str = re.sub(r'\bno+\b', 'no', data_str)
    data_str = re.sub(r'\bcoo+\b', 'cool', data_str)
    data_str = re.sub(r'rt\b', '', data_str)
    data_str = data_str.strip()
    return data_str
```

- remove irrelevant features

```
# remove irrelevant features
def remove_features(data_str):
    # compile regex
    url_re = re.compile('https?://(www.)?\w+\.\w+(/\w+)*/?')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\\d+)')
    mention_re = re.compile('@(\w+)')
    alpha_num_re = re.compile("^[a-z0-9_.]+$")
    # convert to lowercase
    data_str = data_str.lower()
    # remove hyperlinks
    data_str = url_re.sub(' ', data_str)
    # remove @mentions
    data_str = mention_re.sub(' ', data_str)
    # remove puncuation
    data_str = punc_re.sub(' ', data_str)
    # remove numeric 'words'
    data_str = num_re.sub(' ', data_str)
    # remove non a-z 0-9 characters and words shorter than 1 characters
    list_pos = 0
    cleaned_str = ''
    for word in data_str.split():
        if list_pos == 0:
            if alpha_num_re.match(word) and len(word) > 1:
                cleaned_str = word
            else:
                cleaned_str = ' '
        else:
            if alpha_num_re.match(word) and len(word) > 1:
                cleaned_str = cleaned_str + ' ' + word
            else:
                cleaned_str += ' '
        list_pos += 1
    # remove unwanted space, *.split() will automatically split on
    # whitespace and discard duplicates, the " ".join() joins the
    # resulting list into one string.
    return " ".join(cleaned_str.split())
```

- removes stop words

```
# removes stop words
def remove_stops(data_str):
    # expects a string
```

```
    stops = set(stopwords.words("english"))
    list_pos = 0
    cleaned_str = ''
    text = data_str.split()
    for word in text:
        if word not in stops:
            # rebuild cleaned_str
            if list_pos == 0:
                cleaned_str = word
            else:
                cleaned_str = cleaned_str + ' ' + word
            list_pos += 1
    return cleaned_str
```

- Part-of-Speech Tagging

```
# Part-of-Speech Tagging
def tag_and_remove(data_str):
    cleaned_str = ' '
    # noun tags
    nn_tags = ['NN', 'NNP', 'NNP', 'NNPS', 'NNS']
    # adjectives
    jj_tags = ['JJ', 'JJR', 'JJS']
    # verbs
    vb_tags = ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']
    nltk_tags = nn_tags + jj_tags + vb_tags

    # break string into 'words'
    text = data_str.split()

    # tag the text and keep only those with the right tags
    tagged_text = pos_tag(text)
    for tagged_word in tagged_text:
        if tagged_word[1] in nltk_tags:
            cleaned_str += tagged_word[0] + ' '

    return cleaned_str
```

- lemmatization

```
# lemmatization
def lemmatize(data_str):
    # expects a string
    list_pos = 0
    cleaned_str = ''
    lmtzr = WordNetLemmatizer()
    text = data_str.split()
    tagged_words = pos_tag(text)
    for word in tagged_words:
        if 'v' in word[1].lower():
            lemma = lmtzr.lemmatize(word[0], pos='v')
        else:
            lemma = lmtzr.lemmatize(word[0], pos='n')
        if list_pos == 0:
            cleaned_str = lemma
        else:
            cleaned_str = cleaned_str + ' ' + lemma
        list_pos += 1
```

```
    return cleaned_str
```

- setup pyspark udf function

```
# setup pyspark udf function
strip_non_ascii_udf = udf(strip_non_ascii, StringType())
check_blanks_udf = udf(check_blanks, StringType())
check_lang_udf = udf(check_lang, StringType())
fix_abbreviation_udf = udf(fix_abbreviation, StringType())
remove_stops_udf = udf(remove_stops, StringType())
remove_features_udf = udf(remove_features, StringType())
tag_and_remove_udf = udf(tag_and_remove, StringType())
lemmatize_udf = udf(lemmatize, StringType())
```

1. Text processing

- correct the data schema

```
rawdata = rawdata.withColumn('rating', rawdata.rating.cast('float'))

rawdata.printSchema()

root
|-- id: string (nullable = true)
|-- airline: string (nullable = true)
|-- date: string (nullable = true)
|-- location: string (nullable = true)
|-- rating: float (nullable = true)
|-- cabin: string (nullable = true)
|-- value: string (nullable = true)
|-- recommended: string (nullable = true)
|-- review: string (nullable = true)

from datetime import datetime
from pyspark.sql.functions import col

# https://docs.python.org/2/library/datetime.html#strftime-and-strptime-behavior
# 21-Jun-14 <----> %d-%b-%y
to_date =  udf (lambda x: datetime.strptime(x, '%d-%b-%y'), DateType())

rawdata = rawdata.withColumn('date', to_date(col('date')))

rawdata.printSchema()

root
 |-- id: string (nullable = true)
 |-- airline: string (nullable = true)
 |-- date: date (nullable = true)
 |-- location: string (nullable = true)
 |-- rating: float (nullable = true)
 |-- cabin: string (nullable = true)
 |-- value: string (nullable = true)
 |-- recommended: string (nullable = true)
 |-- review: string (nullable = true)

rawdata.show(5)
```

```
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Fli
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Fli
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Wor
+-----+------------------+----------+--------+------+--------+-----+-----------+---
only showing top 5 rows

rawdata = rawdata.withColumn('non_asci', strip_non_ascii_udf(rawdata['review']))


+-----+------------------+----------+--------+------+--------+-----+-----------+---
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Fli
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Fli
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Wor
+-----+------------------+----------+--------+------+--------+-----+-----------+---
only showing top 5 rows

rawdata = rawdata.select(raw_cols+['non_asci'])\
                 .withColumn('fixed_abbrev',fix_abbreviation_udf(rawdata['non_asci'

+-----+------------------+----------+--------+------+--------+-----+-----------+---
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Fli
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Fli
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Wor
+-----+------------------+----------+--------+------+--------+-----+-----------+---
only showing top 5 rows

 rawdata = rawdata.select(raw_cols+['fixed_abbrev'])\
                 .withColumn('stop_text',remove_stops_udf(rawdata['fixed_abbrev']))

+-----+------------------+----------+--------+------+--------+-----+-----------+---
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Fli
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Fli
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Wor
+-----+------------------+----------+--------+------+--------+-----+-----------+---
only showing top 5 rows

rawdata = rawdata.select(raw_cols+['stop_text'])\
                 .withColumn('feat_text',remove_features_udf(rawdata['stop_text']))

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

```
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Fli
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Fli
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Wor
+-----+------------------+----------+--------+------+--------+-----+-----------+---
only showing top 5 rows
```

```
 rawdata = rawdata.select(raw_cols+['feat_text'])\
                  .withColumn('tagged_text',tag_and_remove_udf(rawdata['feat_text']
```

```
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Fli
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Fli
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Wor
+-----+------------------+----------+--------+------+--------+-----+-----------+---
only showing top 5 rows
```

```
 rawdata = rawdata.select(raw_cols+['tagged_text']) \
                  .withColumn('lemm_text',lemmatize_udf(rawdata['tagged_text'])
```

```
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Fli
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Fli
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Wor
+-----+------------------+----------+--------+------+--------+-----+-----------+---
only showing top 5 rows
```

```
 rawdata = rawdata.select(raw_cols+['lemm_text']) \
                  .withColumn("is_blank", check_blanks_udf(rawdata["lemm_text"]))
```

```
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+---
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Fli
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Fli
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Wor
+-----+------------------+----------+--------+------+--------+-----+-----------+---
only showing top 5 rows
```

```
from pyspark.sql.functions import monotonically_increasing_id
# Create Unique ID
rawdata = rawdata.withColumn("uid", monotonically_increasing_id())
```

```
data = rawdata.filter(rawdata["is_blank"] == "False")
```

```
+-----+-----------------+----------+--------+------+--------+-----+-----------+---
|   id|          airline|      date|location|rating|   cabin|value|recommended|
+-----+-----------------+----------+--------+------+--------+-----+-----------+---
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Fli
|10298|       US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Fli
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm
|10134|  Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP
|10912|  United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Wor
+-----+-----------------+----------+--------+------+--------+-----+-----------+---
only showing top 5 rows
```

Pipeline for LDA model

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml import Pipeline
from pyspark.ml.classification import NaiveBayes, RandomForestClassifier
from pyspark.ml.clustering import LDA
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.feature import CountVectorizer

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, an
tokenizer = Tokenizer(inputCol="lemm_text", outputCol="words")
#data = tokenizer.transform(data)
vectorizer = CountVectorizer(inputCol= "words", outputCol="rawFeatures")
idf = IDF(inputCol="rawFeatures", outputCol="features")
#idfModel = idf.fit(data)

lda = LDA(k=20, seed=1, optimizer="em")

pipeline = Pipeline(stages=[tokenizer, vectorizer,idf, lda])


model = pipeline.fit(data)
```

1. Results presentation

- Topics

```
+-----+--------------------+--------------------+
|topic|         termIndices|         termWeights|
+-----+--------------------+--------------------+
|    0|[60, 7, 12, 483, ...|[0.01349507958269...|
|    1|[363, 29, 187, 55...|[0.01247250144447...|
|    2|[46, 107, 672, 27...|[0.01188684264641...|
|    3|[76, 43, 285, 152...|[0.01132638300115...|
|    4|[201, 13, 372, 69...|[0.01337529863256...|
|    5|[122, 103, 181, 4...|[0.00930415977117...|
|    6|[14, 270, 18, 74,...|[0.01253817708163...|
|    7|[111, 36, 341, 10...|[0.01269584954257...|
|    8|[477, 266, 297, 1...|[0.01017486869509...|
|    9|[10, 73, 46, 1, 2...|[0.01050875237546...|
|   10|[57, 29, 411, 10,...|[0.01777350667863...|
```

```
|   11|[293, 119, 385, 4...|[0.01280305149305...|
|   12|[116, 218, 256, 1...|[0.01570714218509...|
|   13|[433, 171, 176, 3...|[0.00819684813575...|
|   14|[74, 84, 45, 108,...|[0.01700630002172...|
|   15|[669, 215, 14, 58...|[0.00779310974971...|
|   16|[198, 21, 98, 164...|[0.01030577084202...|
|   17|[96, 29, 569, 444...|[0.01297142577633...|
|   18|[18, 60, 140, 64,...|[0.01306356985169...|
|   19|[33, 178, 95, 2, ...|[0.00907425683229...|
+-----+--------------------+--------------------+
```

- Topic terms

```
from pyspark.sql.types import ArrayType, StringType

def termsIdx2Term(vocabulary):
    def termsIdx2Term(termIndices):
        return [vocabulary[int(index)] for index in termIndices]
    return udf(termsIdx2Term, ArrayType(StringType()))

vectorizerModel = model.stages[1]
vocabList = vectorizerModel.vocabulary
final = ldatopics.withColumn("Terms", termsIdx2Term(vocabList)("termIndices"))
```

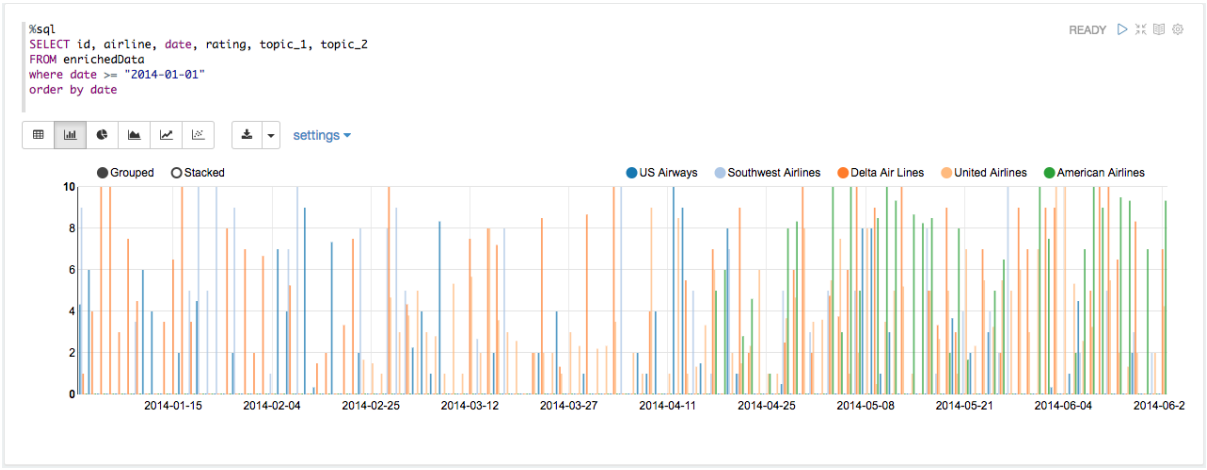
```
+-----+-----------------------------------------------+-------------------------
|topic|termIndices                                    |Terms
+-----+-----------------------------------------------+-------------------------
|0    |[60, 7, 12, 483, 292, 326, 88, 4, 808, 32]     |[pm, plane, board, kid, onl
|1    |[363, 29, 187, 55, 48, 647, 30, 9, 204, 457]   |[dublin, class, th, sit, en
|2    |[46, 107, 672, 274, 92, 539, 23, 27, 279, 8]   |[economy, sfo, milwaukee, d
|3    |[76, 43, 285, 152, 102, 34, 300, 113, 24, 31]  |[didn, pay, lose, different
|4    |[201, 13, 372, 692, 248, 62, 211, 187, 105, 110]|[houston, crew, heathrow, l
|5    |[122, 103, 181, 48, 434, 10, 121, 147, 934, 169]|[lhr, serve, screen, entert
|6    |[14, 270, 18, 74, 70, 37, 16, 450, 3, 20]      |[check, employee, gate, lin
|7    |[111, 36, 341, 10, 320, 528, 844, 19, 195, 524]|[atlanta, first, toilet, de
|8    |[477, 266, 297, 185, 1, 33, 22, 783, 17, 908]  |[fuel, group, pas, boarding
|9    |[10, 73, 46, 1, 248, 302, 213, 659, 48, 228]   |[delta, lax, economy, seat,
|10   |[57, 29, 411, 10, 221, 121, 661, 19, 805, 733] |[business, class, fra, delt
|11   |[293, 119, 385, 481, 503, 69, 13, 87, 176, 545]|[march, ua, manchester, phx
|12   |[116, 218, 256, 156, 639, 20, 365, 18, 22, 136]|[san, clt, francisco, secon
|13   |[433, 171, 176, 339, 429, 575, 10, 26, 474, 796]|[daughter, small, aa, ba, s
|14   |[74, 84, 45, 108, 342, 111, 315, 87, 52, 4]    |[line, agent, next, hotel,
|15   |[669, 215, 14, 58, 561, 59, 125, 179, 93, 5]   |[fit, carry, check, people,
|16   |[198, 21, 98, 164, 57, 141, 345, 62, 121, 174] |[ife, good, nice, much, bus
|17   |[96, 29, 569, 444, 15, 568, 21, 103, 657, 505] |[phl, class, diego, lady, f
|18   |[18, 60, 140, 64, 47, 40, 31, 35, 2, 123]      |[gate, pm, phoenix, connect
|19   |[33, 178, 95, 2, 9, 284, 42, 4, 89, 31]        |[trip, counter, philadelphi
+-----+-----------------------------------------------+-------------------------
```

- LDA results

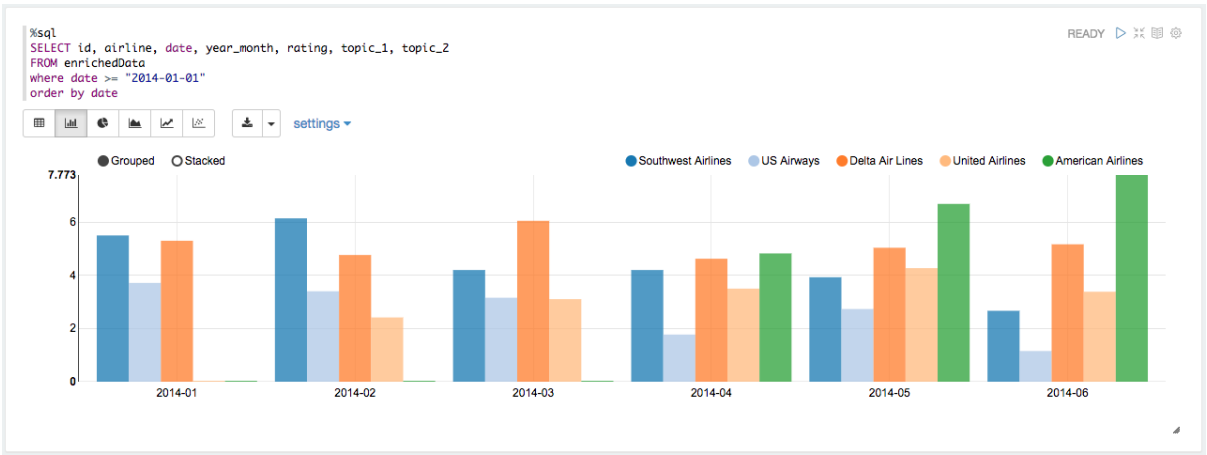
```
+-----+-----------------+----------+-----------+------+--------------------+------
|   id|          airline|      date|      cabin|rating|               words|
+-----+-----------------+----------+-----------+------+--------------------+------
|10551|Southwest Airlines|2013-11-06|   Business|   1.0|[flight, chicago,...|(4695,
|10298|        US Airways|2014-03-31|   Business|   1.0|[flight, manchest...|(4695,
|10564|Southwest Airlines|2013-09-06|    Economy|  10.0|[executive, plati...|(4695,
|10134|   Delta Air Lines|2013-12-10|    Economy|   8.0|[msp, jfk, mxp, r...|(4695,
```

```
|10912|   United Airlines|2014-04-07|    Economy|   3.0|[worst, airline, ...|(4695,
|10089|   Delta Air Lines|2014-02-18|    Economy|   2.0|[dl, mia, lax, im...|(4695,
|10385|        US Airways|2013-10-21|    Economy|  10.0|[flew, gla, phl, ...|(4695,
|10249|        US Airways|2014-06-17|    Economy|   1.0|[friend, book, fl...|(4695,
|10289|        US Airways|2014-04-12|    Economy|  10.0|[flew, air, rome,...|(4695,
|10654|Southwest Airlines|2012-07-10|    Economy|   8.0|[lhr, jfk, think,...|(4695,
|10754| American Airlines|2014-05-04|    Economy|  10.0|[san, diego, moli...|(4695,
|10646|Southwest Airlines|2012-08-17|    Economy|   7.0|[toledo, co, stop...|(4695,
|10097|   Delta Air Lines|2014-02-03|First Class|  10.0|[honolulu, la, fi...|(4695,
|10132|   Delta Air Lines|2013-12-16|    Economy|   7.0|[manchester, uk, ...|(4695,
|10560|Southwest Airlines|2013-09-20|    Economy|   9.0|[first, time, sou...|(4695,
|10579|Southwest Airlines|2013-07-25|    Economy|   0.0|[plane, land, pm,...|(4695,
|10425|        US Airways|2013-08-06|    Economy|   3.0|[airway, bad, pro...|(4695,
|10650|Southwest Airlines|2012-07-27|    Economy|   9.0|[flew, jfk, lhr, ...|(4695,
|10260|        US Airways|2014-06-03|    Economy|   1.0|[february, air, u...|(4695,
|10202|   Delta Air Lines|2013-09-14|    Economy|  10.0|[aug, lhr, jfk, b...|(4695,
+-----+------------------+----------+-----------+------+--------------------+------
only showing top 20 rows
```

- Average rating and airlines for each day



- Average rating and airlines for each month

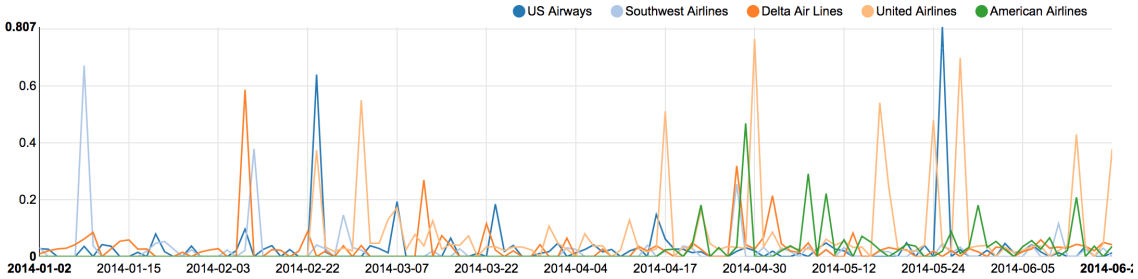


- Topic 1 corresponding to time line
- reviews (documents) relate to topic 1

```
%sql
SELECT id, airline, date, year_month, rating, topic_1, topic_2
FROM enrichedData
where date >= "2014-01-01"
order by date
```

READY

settings



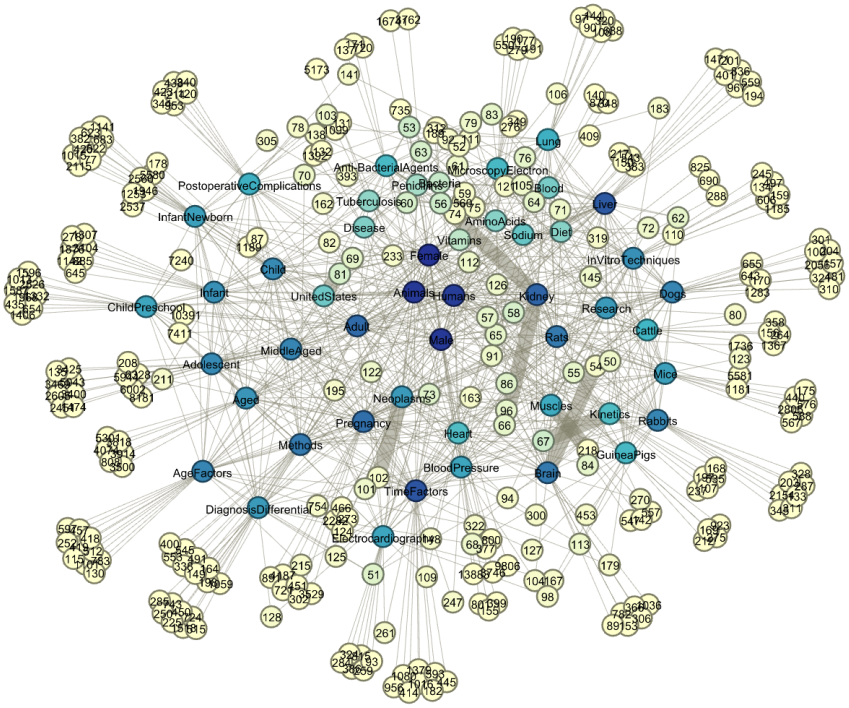
id	airline	date	review
10263	US Airways	2014-05-25	"Delays on all booked flights. Outward bound - Dublin to Philadelphia Philadelphia to Vegas. Vegas to LA. Baggage did not m arrived some hours later. Cabin staff were unfriendly and quite rude. From Dublin We were sitting at the back of the plane and and we were told ""thats what you get when you travel at the back"" . I opened my little tub of butter which had been heated in hot liquid and went all over my hand. I said to the stewardess who was passing by who could have brought me a napkin to sped by saying ""oh I know that happens"". Only that I had a tray of food on my lap she would have had more to deal with! W baggage was to be stored in the overhead lockers or underneath the seat in front. This obviously does not apply to cabin staf back centre aisle seats of the plane it was not secured in any way. Their baggage was a danger to all the passengers in that a what they preach and put staff baggage in the hold. Cabin crew were ungroomed in appearance. Homeward bound to Dublin delayed resulting in us overnighting in Orlando very grateful for the overnight accommodation provided at the Hyatt. Flew to (Boarding for Dublin at Charlotte was very confused and inefficiently exercised by Gate staff - resulting in delay in take-off. En quality on those flights that had the facility. General impression overall. Very Disappointing."

CHAPTER

FOURTEEN

SOCIAL NETWORK ANALYSIS

Note: A Touch of Cloth,linked in countless ways. – old Chinese proverb



14.1 Introduction

14.2 Co-occurrence Network

Co-occurrence networks are generally used to provide a graphic visualization of potential relationships between people, organizations, concepts or other entities represented within written material. The generation and visualization of co-occurrence networks has become practical with the advent of electronically stored text amenable to text mining.

14.2.1 Methodology

- Build Corpus C
- Build Document-Term matrix D based on Corpus C
- Compute Term-Document matrix D^T
- Adjacency Matrix $A = D^T \cdot D$

There are four main components in this algorithm in the algorithm: Corpus C, Document-Term matrix D, Term-Document matrix D^T and Adjacency Matrix A. In this demo part, I will show how to build those four main components.

Given that we have three groups of friends, they are

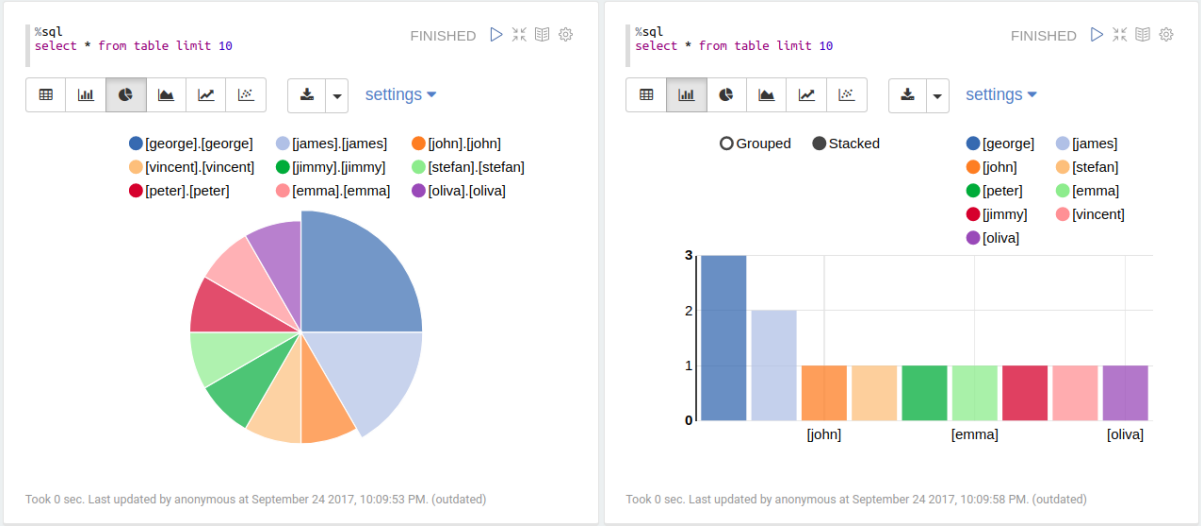
```
+-----------------------------------+
|words                              |
+-----------------------------------+
|[[george] [jimmy] [john] [peter]]  |
|[[vincent] [george] [stefan] [james]]|
|[[emma] [james] [olivia] [george]] |
+-----------------------------------+
```

1. Corpus C

Then we can build the following corpus based on the unique elements in the given group data:

```
[u'george', u'james', u'jimmy', u'peter', u'stefan', u'vincent', u'olivia', u'john'
```

The corresponding elements frequency:



2. Document-Term matrix D based on Corpus C (CountVectorizer)

```
from pyspark.ml.feature import CountVectorizer
count_vectorizer_wo = CountVectorizer(inputCol='term', outputCol='features')
# with total unique vocabulary
countVectorizer_mod_wo = count_vectorizer_wo.fit(df)
countVectorizer_twitter_wo = countVectorizer_mod_wo.transform(df)
# with truncated unique vocabulary (99%)
count_vectorizer = CountVectorizer(vocabSize=48,inputCol='term',outputCol='features
```

```
countVectorizer_mod = count_vectorizer.fit(df)
countVectorizer_twitter = countVectorizer_mod.transform(df)

+-------------------------------+
|features                       |
+-------------------------------+
|(9,[0,2,3,7],[1.0,1.0,1.0,1.0])|
|(9,[0,1,4,5],[1.0,1.0,1.0,1.0])|
|(9,[0,1,6,8],[1.0,1.0,1.0,1.0])|
+-------------------------------+
```

- Term-Document matrix D^T

  RDD:

```
[array([ 1.,  1.,  1.]), array([ 0.,  1.,  1.]), array([ 1.,  0.,  0.]),
 array([ 1.,  0.,  0.]), array([ 0.,  1.,  0.]), array([ 0.,  1.,  0.]),
 array([ 0.,  0.,  1.]), array([ 1.,  0.,  0.]), array([ 0.,  0.,  1.])]
```

  Matrix:

```
array([[ 1.,  1.,  1.],
       [ 0.,  1.,  1.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.],
       [ 1.,  0.,  0.],
       [ 0.,  0.,  1.]])
```

3. Adjacency Matrix $A = D^T \cdot D$

   RDD:

```
[array([ 1.,  1.,  1.]), array([ 0.,  1.,  1.]), array([ 1.,  0.,  0.]),
 array([ 1.,  0.,  0.]), array([ 0.,  1.,  0.]), array([ 0.,  1.,  0.]),
 array([ 0.,  0.,  1.]), array([ 1.,  0.,  0.]), array([ 0.,  0.,  1.])]
```

   Matrix:

```
array([[ 3.,  2.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 2.,  2.,  0.,  0.,  1.,  1.,  1.,  0.,  1.],
       [ 1.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.],
       [ 1.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.],
       [ 1.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  1.],
       [ 1.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.],
       [ 1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  1.]])
```

14.2.2 Coding Puzzle from my interview

- Problem

The attached utf-8 encoded text file contains the tags associated with an online biomedical scientific article formatted as follows (size: 100000). Each Scientific article is represented by a line in the file delimited by carriage return.

```
+--------------------+
|               words|
+--------------------+
|[ACTH Syndrome, E...|
|[Antibody Formati...|
|[Adaptation, Phys...|
|[Aerosol Propella...|
+--------------------+
only showing top 4 rows
```

Write a program that, using this file as input, produces a list of pairs of tags which appear TOGETHER in any order and position in at least fifty different Scientific articles. For example, in the above sample, [Female] and [Humans] appear together twice, but every other pair appears only once. Your program should output the pair list to stdout in the same form as the input (eg tag 1, tag 2n).

- My solution

  The corresponding words frequency:

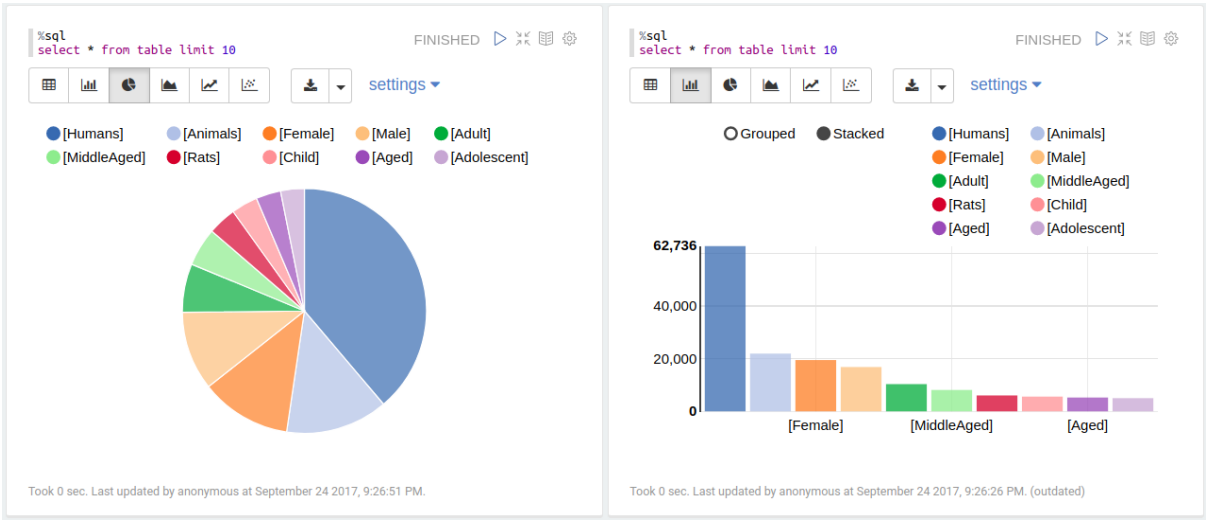


Figure 14.1: Word frequency

Output:

```
+----------+------+-------+
|    term.x|term.y|   freq|
+----------+------+-------+
|    Female|Humans|16741.0|
|      Male|Humans|13883.0|
|     Adult|Humans|10391.0|
|      Male|Female| 9806.0|
|MiddleAged|Humans| 8181.0|
|     Adult|Female| 7411.0|
|     Adult|  Male| 7240.0|
|MiddleAged|  Male| 6328.0|
|MiddleAged|Female| 6002.0|
|MiddleAged| Adult| 5944.0|
+----------+------+-------+
only showing top 10 rows
```

The corresponding Co-occurrence network:

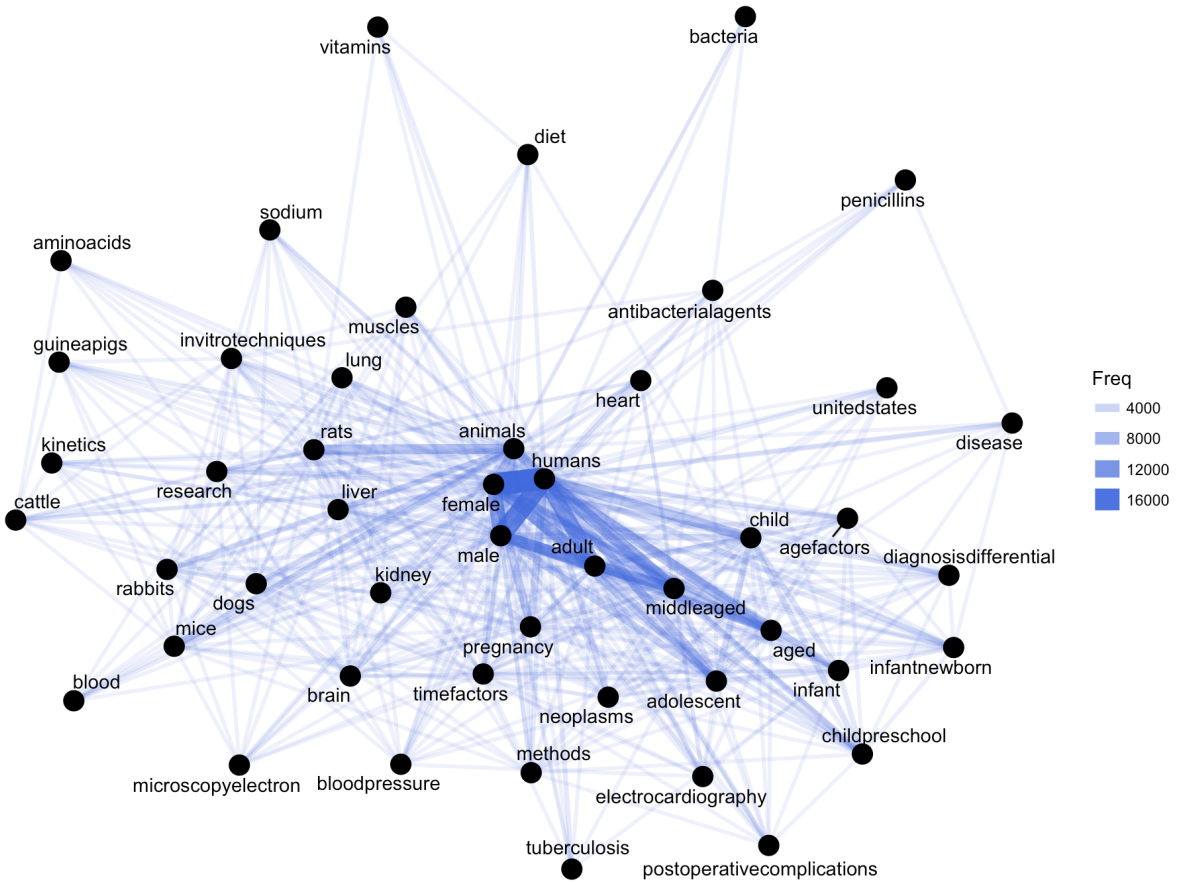


Figure 14.2: Co-occurrence network

Then you will get Figure *Co-occurrence network*

14.3 Appendix: matrix multiplication in PySpark

1. load test matrix

```
df = spark.read.csv("matrix1.txt",sep=",",inferSchema=True)
df.show()
```

```
+---+---+---+---+
|_c0|_c1|_c2|_c3|
+---+---+---+---+
|1.2|3.4|2.3|1.1|
|2.3|1.1|1.5|2.2|
|3.3|1.8|4.5|3.3|
|5.3|2.2|4.5|4.4|
|9.3|8.1|0.3|5.5|
|4.5|4.3|2.1|6.6|
+---+---+---+---+
```

2. main function for matrix multiplication in PySpark

```
from pyspark.sql import functions as F
from functools import reduce
# reference: https://stackoverflow.com/questions/44348527/matrix-multiplication-at-a-in
# do the sum of the multiplication that we want, and get
# one data frame for each column
colDFs = []
for c2 in df.columns:
    colDFs.append( df.select( [ F.sum(df[c1]*df[c2]).alias("op_{0}".format(i)) for i,c1
# now union those separate data frames to build the "matrix"
mtxDF = reduce(lambda a,b: a.select(a.columns).union(b.select(a.columns)), colDFs )
mtxDF.show()
```

```
+------------------+------------------+------------------+------------------+
|              op_0|              op_1|              op_2|              op_3|
+------------------+------------------+------------------+------------------+
|            152.45|118.88999999999999|             57.15|121.44000000000001|
|118.88999999999999|104.94999999999999|             38.93|             94.71|
|             57.15|             38.93|52.540000000000006|             55.99|
|121.44000000000001|             94.71|             55.99|110.10999999999999|
+------------------+------------------+------------------+------------------+
```

3. Validation with python version

```
import numpy as np
a = np.genfromtxt("matrix1.txt",delimiter=",")
np.dot(a.T, a)
```

```
array([[152.45, 118.89,  57.15, 121.44],
       [118.89, 104.95,  38.93,  94.71],
       [ 57.15,  38.93,  52.54,  55.99],
       [121.44,  94.71,  55.99, 110.11]])
```

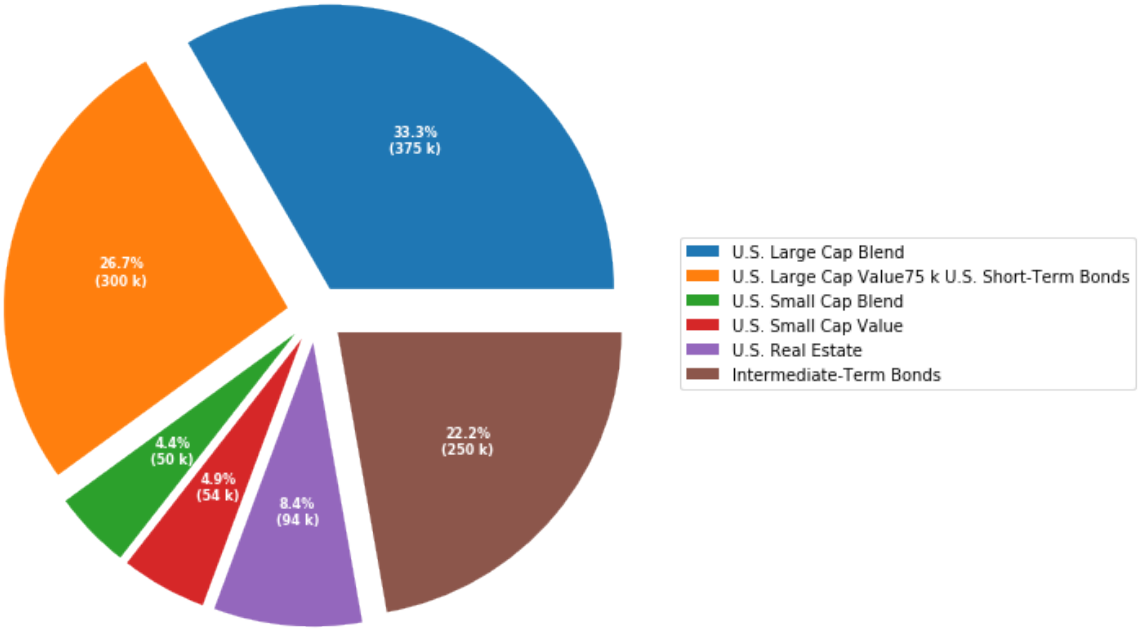
14.4 Correlation Network

TODO ..

CHAPTER

FIFTEEN

ALS: STOCK PORTFOLIO RECOMMENDATIONS



Code for the above figure:

```
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 8), subplot_kw=dict(aspect="equal"))

recipe = ["375 k U.S. Large Cap Blend",
          "300 k U.S. Large Cap Value"
          "75 k U.S. Short-Term Bonds",
          "50 k U.S. Small Cap Blend",
          "55 k U.S. Small Cap Value",
          "95 k U.S. Real Estate",
          "250 k Intermediate-Term Bonds"]

data = [float(x.split()[0]) for x in recipe]
ingredients = [' '.join(x.split()[2:]) for x in recipe]

print(data)
print(ingredients)
```

```
def func(pct, allvals):
    absolute = int(pct/100.*np.sum(allvals))
    return "{:.1f}%\n({:d} k)".format(pct, absolute)

explode = np.empty(len(data))#(0.1, 0.1, 0.1,  0.1, 0.1, 0.1)  # explode 1st slice
explode.fill(0.1)

wedges, texts, autotexts = ax.pie(data, explode=explode, autopct=lambda pct: func(pct, d
                                  textprops=dict(color="w"))
ax.legend(wedges, ingredients,
          #title="Stock portfolio",
          loc="center left",
          bbox_to_anchor=(1, 0, 0.5, 1))

plt.setp(autotexts, size=8, weight="bold")

#ax.set_title("Stock portfolio")

plt.show()
```

15.1 Recommender systems

Recommender systems or recommendation systems (sometimes replacing "system" with a synonym such as platform or engine) are a subclass of information filtering system that seek to predict the "rating" or "preference" that a user would give to an item."

The main idea is to build a matrix users `R` items rating values and try to factorize it, to recommend main products rated by other users. A popular approach for this is matrix factorization is Alternating Least Squares (ALS)

15.2 Alternating Least Squares

Apache Spark ML implements ALS for collaborative filtering, a very popular algorithm for making recommendations.

ALS recommender is a matrix factorization algorithm that uses Alternating Least Squares with Weighted-Lamda-Regularization (ALS-WR). It factors the user to item matrix `A` into the user-to-feature matrix `U` and the item-to-feature matrix `M`: It runs the ALS algorithm in a parallel fashion. The ALS algorithm should uncover the latent factors that explain the observed user to item ratings and tries to find optimal factor weights to minimize the least squares between predicted and actual ratings.

https://www.elenacuoco.com/2016/12/22/alternating-least-squares-als-spark-ml/

15.3 Demo

- The Jupyter notebook can be download from ALS Recommender systems.
- The data can be downloaf from German Credit.

15.3.1 Load and clean data

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark RFM example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
df_raw = spark.read.format('com.databricks.spark.csv').\
                        options(header='true', \
                        inferschema='true').\
            load("Online Retail.csv",header=True);
```

check the data set

```
df_raw.show(5)
df_raw.printSchema()
```

Then you will get

```
+---------+---------+--------------------+--------+-----------+---------+----------+---
|InvoiceNo|StockCode|         Description|Quantity| InvoiceDate|UnitPrice|CustomerID|
+---------+---------+--------------------+--------+-----------+---------+----------+---
|   536365|   85123A|WHITE HANGING HEA...|       6|12/1/10 8:26|     2.55|     17850|Uni
|   536365|    71053| WHITE METAL LANTERN|       6|12/1/10 8:26|     3.39|     17850|Uni
|   536365|   84406B|CREAM CUPID HEART...|       8|12/1/10 8:26|     2.75|     17850|Uni
|   536365|   84029G|KNITTED UNION FLA...|       6|12/1/10 8:26|     3.39|     17850|Uni
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|12/1/10 8:26|     3.39|     17850|Uni
+---------+---------+--------------------+--------+-----------+---------+----------+---
only showing top 5 rows

root
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: string (nullable = true)
 |-- UnitPrice: double (nullable = true)
 |-- CustomerID: integer (nullable = true)
 |-- Country: string (nullable = true)
```

3. Data clean and data manipulation

- check and remove the `null` values

```
from pyspark.sql.functions import count

def my_count(df_in):
    df_in.agg( *[ count(c).alias(c) for c in df_in.columns ] ).show()

import pyspark.sql.functions as F
from pyspark.sql.functions import round
df_raw = df_raw.withColumn('Asset',round( F.col('Quantity') * F.col('UnitPrice'), 2 ))
```

```
df = df_raw.withColumnRenamed('StockCode', 'Cusip')\
           .select('CustomerID','Cusip','Quantity','UnitPrice','Asset')

my_count(df)
```

```
+----------+------+--------+---------+------+
|CustomerID| Cusip|Quantity|UnitPrice| Asset|
+----------+------+--------+---------+------+
|    406829|541909|  541909|   541909|541909|
+----------+------+--------+---------+------+
```

Since the count results are not the same, we have some null value in the `CustomerID` column. We can drop these records from the dataset.

```
df =  df.filter(F.col('Asset')>=0)
df = df.dropna(how='any')
my_count(df)
```

```
+----------+------+--------+---------+------+
|CustomerID| Cusip|Quantity|UnitPrice| Asset|
+----------+------+--------+---------+------+
|    397924|397924|  397924|   397924|397924|
+----------+------+--------+---------+------+
```

```
df.show(3)
```

```
+----------+------+--------+---------+-----+
|CustomerID| Cusip|Quantity|UnitPrice|Asset|
+----------+------+--------+---------+-----+
|     17850|85123A|       6|     2.55| 15.3|
|     17850| 71053|       6|     3.39|20.34|
|     17850|84406B|       8|     2.75| 22.0|
+----------+------+--------+---------+-----+
only showing top 3 rows
```

- Convert the `Cusip` to consistent format

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType

def toUpper(s):
    return s.upper()

upper_udf = udf(lambda x: toUpper(x), StringType())
```

- Find the most top `n` stockes

```
pop = df.groupBy('Cusip')\
  .agg(F.count('CustomerID').alias('Customers'),F.round(F.sum('Asset'),2).alias('TotalAs
  .sort([F.col('Customers'),F.col('TotalAsset')],ascending=[0,0])

pop.show(5)
```

```
+------+---------+----------+
| Cusip|Customers|TotalAsset|
+------+---------+----------+
|85123A|     2035|  100603.5|
| 22423|     1724| 142592.95|
```

```
|85099B|     1618|  85220.78|
| 84879|     1408|  56580.34|
| 47566|     1397|  68844.33|
+------+---------+----------+
only showing top 5 rows
```

Build feature matrix ————————————-+

- Fetch the top `n` cusip list

```
top = 10
cusip_lst = pd.DataFrame(pop.select('Cusip').head(top)).astype('str').iloc[:, 0].tolist
cusip_lst.insert(0,'CustomerID')
```

- Create the portfolio table for each customer

```
pivot_tab = df.groupBy('CustomerID').pivot('Cusip').sum('Asset')
pivot_tab = pivot_tab.fillna(0)
```

- Fetch the most `n` stock's portfolio table for each customer

```
selected_tab  = pivot_tab.select(cusip_lst)
selected_tab.show(4)
```

```
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
|CustomerID|85123A|22423|85099B|84879|47566|20725|22720|20727|POST|23203|
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
|     16503|   0.0|  0.0|   0.0|  0.0|  0.0|  0.0|  0.0| 33.0| 0.0|  0.0|
|     15727| 123.9| 25.5|   0.0|  0.0|  0.0| 33.0| 99.0|  0.0| 0.0|  0.0|
|     14570|   0.0|  0.0|   0.0|  0.0|  0.0|  0.0|  0.0|  0.0| 0.0|  0.0|
|     14450|   0.0|  0.0|  8.32|  0.0|  0.0|  0.0| 49.5|  0.0| 0.0|  0.0|
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
only showing top 4 rows
```

- Build the `rating` matrix

```
def elemwiseDiv(df_in):
    num = len(df_in.columns)
    temp = df_in.rdd.map(lambda x: list(flatten([x[0],[x[i]/float(sum(x[1:]))
                                                       if sum(x[1:])>0 else x[i]
                                                       for i in range(1,num)]])))
    return spark.createDataFrame(temp,df_in.columns)

ratings = elemwiseDiv(selected_tab)

ratings.show(4)
```

```
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
|CustomerID|85123A|22423|85099B|84879|47566|20725|22720|20727|POST|23203|
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
|     16503|   0.0|  0.0|   0.0|  0.0|  0.0|  0.0|  0.0|  1.0| 0.0|  0.0|
|     15727|  0.44| 0.09|   0.0|  0.0|  0.0| 0.12| 0.35|  0.0| 0.0|  0.0|
|     14570|   0.0|  0.0|   0.0|  0.0|  0.0|  0.0|  0.0|  0.0| 0.0|  0.0|
|     14450|   0.0|  0.0|  0.14|  0.0|  0.0|  0.0| 0.86|  0.0| 0.0|  0.0|
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
```

- Convert `rating` matrix to long table

```
from pyspark.sql.functions import array, col, explode, struct, lit

def to_long(df, by):
        """
        reference: https://stackoverflow.com/questions/37864222/transpose-column-to-row-
        """

    # Filter dtypes and split into column names and type description
    cols, dtypes = zip(*((c, t) for (c, t) in df.dtypes if c not in by))
    # Spark SQL supports only homogeneous columns
    assert len(set(dtypes)) == 1, "All columns have to be of the same type"

    # Create and explode an array of (column_name, column_value) structs
    kvs = explode(array([
      struct(lit(c).alias("Cusip"), col(c).alias("rating")) for c in cols
    ])).alias("kvs")

df_all = to_long(ratings,['CustomerID'])
df_all.show(5)
```

```
+----------+------+------+
|CustomerID| Cusip|rating|
+----------+------+------+
|     16503|85123A|   0.0|
|     16503| 22423|   0.0|
|     16503|85099B|   0.0|
|     16503| 84879|   0.0|
|     16503| 47566|   0.0|
+----------+------+------+
only showing top 5 rows
```

- Convert the string `Cusip` to numerical index

```
from pyspark.ml.feature import StringIndexer
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='Cusip',
                             outputCol='indexedCusip').fit(df_all)
df_all = labelIndexer.transform(df_all)

df_all.show(5, True)
df_all.printSchema()
```

```
+----------+------+------+------------+
|CustomerID| Cusip|rating|indexedCusip|
+----------+------+------+------------+
|     16503|85123A|   0.0|         6.0|
|     16503| 22423|   0.0|         9.0|
|     16503|85099B|   0.0|         5.0|
|     16503| 84879|   0.0|         1.0|
|     16503| 47566|   0.0|         0.0|
+----------+------+------+------------+
only showing top 5 rows

root
 |-- CustomerID: long (nullable = true)
 |-- Cusip: string (nullable = false)
 |-- rating: double (nullable = true)
 |-- indexedCusip: double (nullable = true)
```

15.3.2 Train model

- build `train` and `test` dataset

```
train, test = df_all.randomSplit([0.8,0.2])

train.show(5)
test.show(5)
```

```
+----------+-----+------------+------------------+
|CustomerID|Cusip|indexedCusip|            rating|
+----------+-----+------------+------------------+
|     12940|20725|         2.0|               0.0|
|     12940|20727|         4.0|               0.0|
|     12940|22423|         9.0|0.49990198000392083|
|     12940|22720|         3.0|               0.0|
|     12940|23203|         7.0|               0.0|
+----------+-----+------------+------------------+
only showing top 5 rows

+----------+-----+------------+------------------+
|CustomerID|Cusip|indexedCusip|            rating|
+----------+-----+------------+------------------+
|     12940|84879|         1.0|0.1325230346990786|
|     13285|20725|         2.0|0.2054154995331466|
|     13285|20727|         4.0|0.2054154995331466|
|     13285|47566|         0.0|               0.0|
|     13623|23203|         7.0|               0.0|
+----------+-----+------------+------------------+
only showing top 5 rows
```

- train model

```
import itertools
from math import sqrt
from operator import add
import sys
from pyspark.ml.recommendation import ALS

from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
def computeRmse(model, data):
    """
    Compute RMSE (Root mean Squared Error).
    """
    predictions = model.transform(data)
    rmse = evaluator.evaluate(predictions)
    print("Root-mean-square error = " + str(rmse))
    return rmse

#train models and evaluate them on the validation set

ranks = [4,5]
lambdas = [0.05]
numIters = [30]
bestModel = None
```

```
bestValidationRmse = float("inf")
bestRank = 0
bestLambda = -1.0
bestNumIter = -1

val = test.na.drop()
for rank, lmbda, numIter in itertools.product(ranks, lambdas, numIters):
    als = ALS(rank=rank, maxIter=numIter, regParam=lmbda, numUserBlocks=10, numItemBlock
              alpha=1.0,
              userCol="CustomerID", itemCol="indexedCusip", seed=1, ratingCol="rating",
    model=als.fit(train)

    validationRmse = computeRmse(model, val)
    print("RMSE (validation) = %f for the model trained with " % validationRmse + \
           "rank = %d, lambda = %.1f, and numIter = %d." % (rank, lmbda, numIter))
    if (validationRmse, bestValidationRmse):
        bestModel = model
        bestValidationRmse = validationRmse
        bestRank = rank
        bestLambda = lmbda
        bestNumIter = numIter

model = bestModel
```

15.3.3 Make prediction

- make prediction

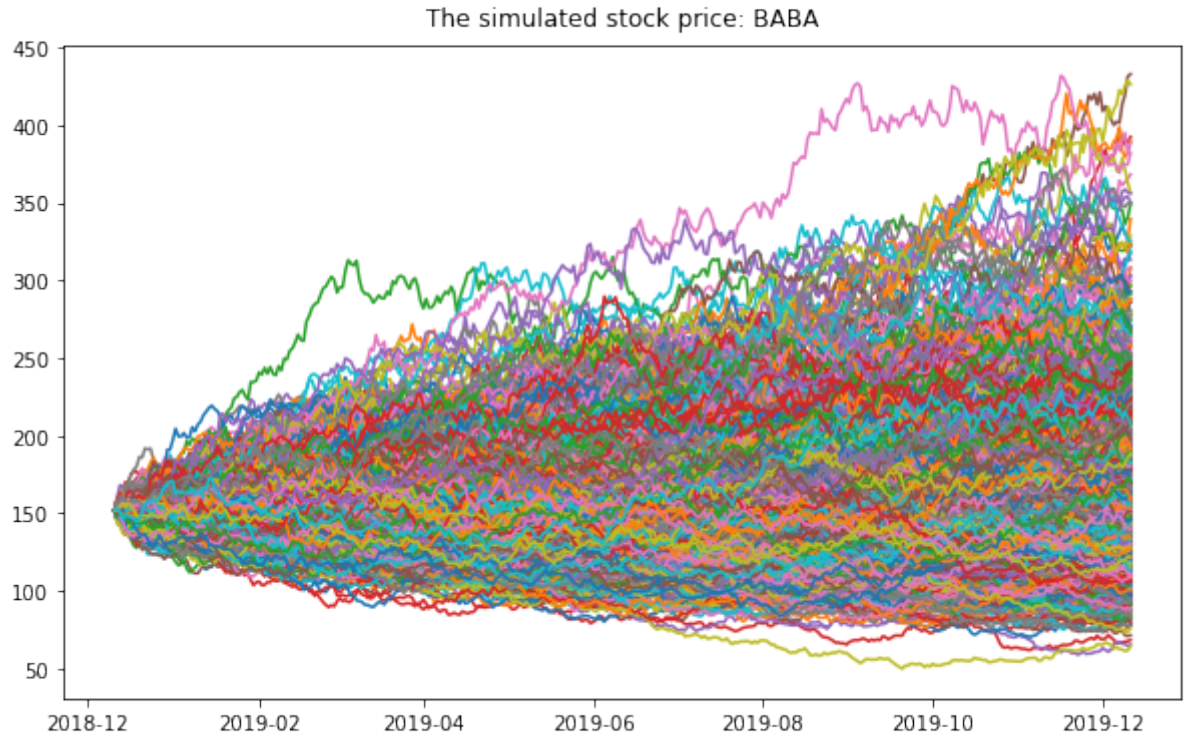
```
topredict=test[test['rating']==0]

predictions=model.transform(topredict)
predictions.filter(predictions.prediction>0)\
           .sort([F.col('CustomerID'),F.col('Cusip')],ascending=[0,0]).show(5)
```

```
+----------+------+------------+------+------------+
|CustomerID| Cusip|indexedCusip|rating|  prediction|
+----------+------+------------+------+------------+
|     18283| 47566|         0.0|   0.0|  0.01625076|
|     18282|85123A|         6.0|   0.0| 0.057172246|
|     18282| 84879|         1.0|   0.0| 0.059531752|
|     18282| 23203|         7.0|   0.0| 0.010502596|
|     18282| 22720|         3.0|   0.0| 0.053893942|
+----------+------+------------+------+------------+
only showing top 5 rows
```

CHAPTER

SIXTEEN

MONTE CARLO SIMULATION



Monte Carlo simulations are just a way of estimating a fixed parameter by repeatedly generating random numbers. More details can be found at A Zero Math Introduction to Markov Chain Monte Carlo Methods.

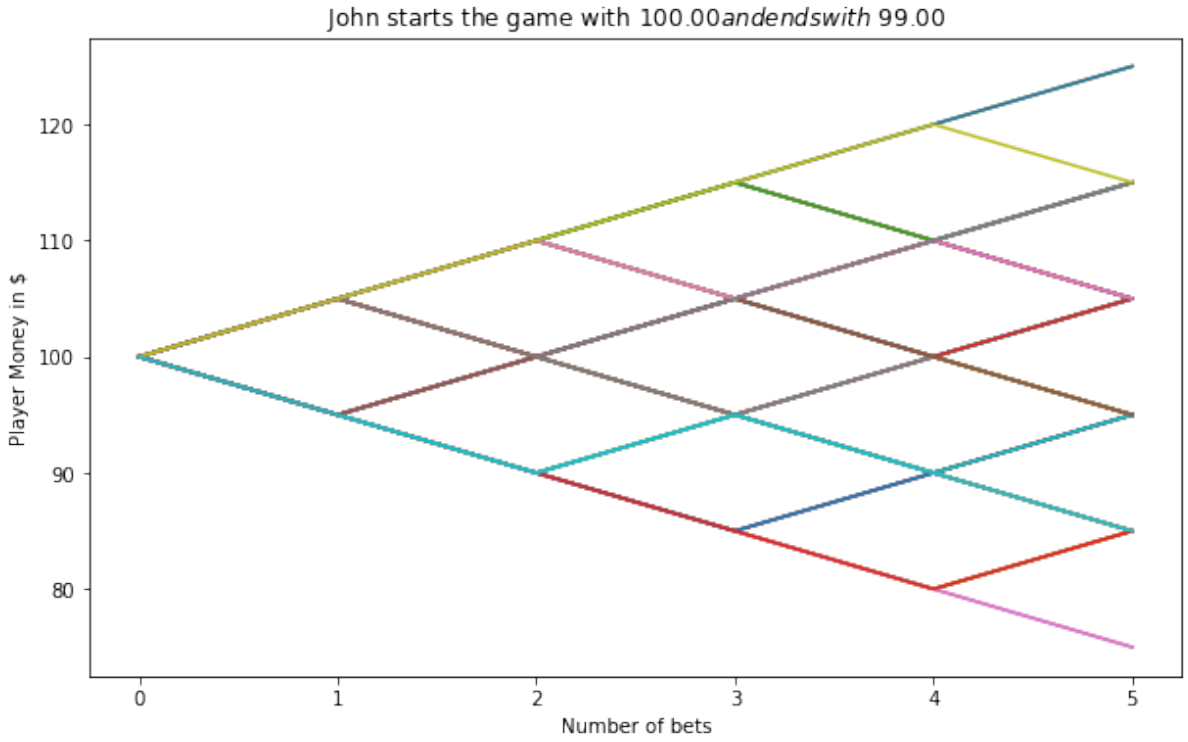
Monte Carlo simulation is a technique used to understand the impact of risk and uncertainty in financial, project management, cost, and other forecasting models. A Monte Carlo simulator helps one visualize most or all of the potential outcomes to have a better idea regarding the risk of a decision. More details can be found at The house always wins.

16.1 Simulating Casino Win

We assume that the player John has the 49% chance to win the game and the wager will be $5 per game.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

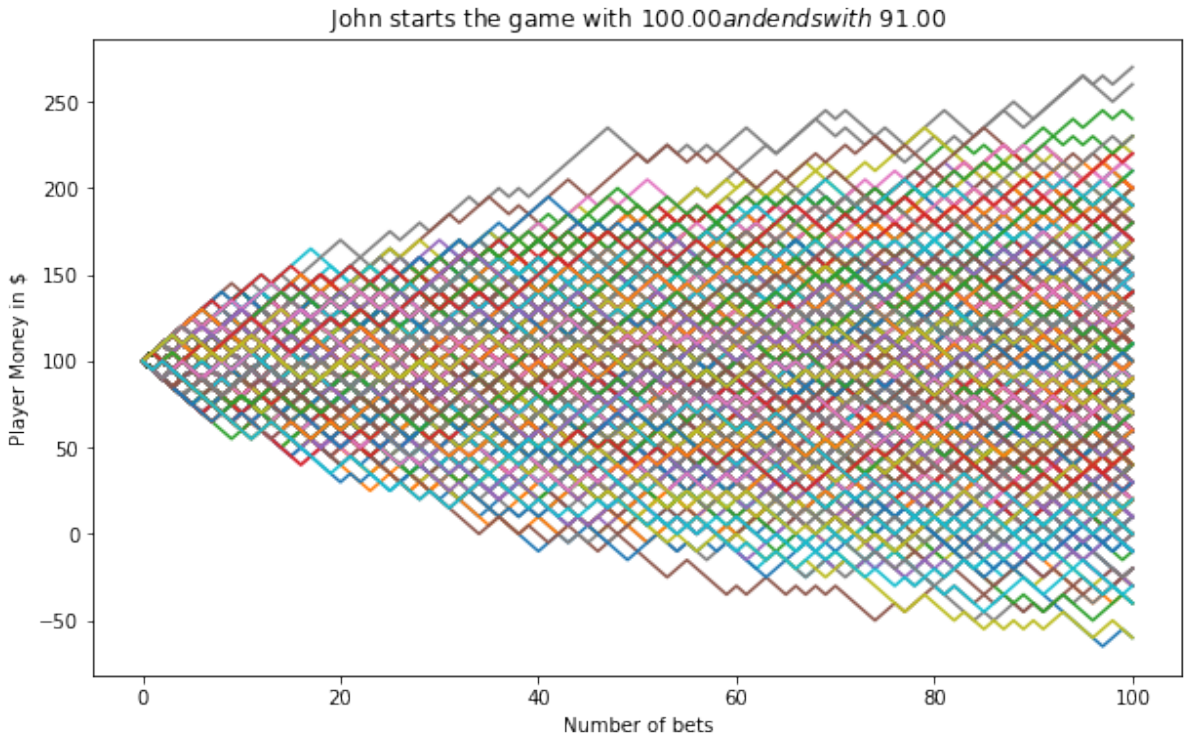
```
start_m =100
wager = 5
bets = 100
trials = 1000

trans = np.vectorize(lambda t: -wager if t <=0.51 else wager)

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(1,1,1)

end_m = []

for i in range(trials):
    money = reduce(lambda c, x: c + [c[-1] + x], trans(np.random.random(bets)), [start_m
    end_m.append(money[-1])
    plt.plot(money)

plt.ylabel('Player Money in $')
plt.xlabel('Number of bets')
plt.title(("John starts the game with $ %.2f and ends with $ %.2f")%(start_m,sum(end_m)
plt.show()
```



16.2 Simulating a Random Walk

16.2.1 Fetch the histrical stock price

1. Fecth the data. If you need the code for this piece, you can contact with me.




```
stock.tail(4)
```

```
+----------+----------+----------+----------+----------+----------+--------+
|      Date|      Open|      High|       Low|     Close| Adj Close|  Volume|
+----------+----------+----------+----------+----------+----------+--------+
|2018-12-07|155.399994|158.050003|151.729996|153.059998|153.059998|17447900|
|2018-12-10|150.389999|152.809998|147.479996|151.429993|151.429993|15525500|
|2018-12-11|155.259995|156.240005|150.899994|151.830002|151.830002|13651900|
|2018-12-12|155.240005|156.169998|151.429993|     151.5|     151.5|16597900|
+----------+----------+----------+----------+----------+----------+--------+
```

2. Convert the `str` type date to date type

```
stock['Date'] = pd.to_datetime(stock['Date'])
```

3. Data visualization

```
# Plot everything by leveraging the very powerful matplotlib package
width = 10
height = 6
data = stock
fig = plt.figure(figsize=(width, height))
ax = fig.add_subplot(1,1,1)
ax.plot(data.Date, data.Close, label='Close')
ax.plot(data.Date, data.High, label='High')
# ax.plot(data.Date, data.Low, label='Low')
ax.set_xlabel('Date')
ax.set_ylabel('price ($)')
ax.legend()
ax.set_title('Stock price: ' + ticker, y=1.01)
#plt.xticks(rotation=70)
plt.show()
# Plot everything by leveraging the very powerful matplotlib package
```

```
fig = plt.figure(figsize=(width, height))
ax = fig.add_subplot(1,1,1)
ax.plot(data.Date, data.Volume, label='Volume')
#ax.plot(data.Date, data.High, label='High')
# ax.plot(data.Date, data.Low, label='Low')
ax.set_xlabel('Date')
ax.set_ylabel('Volume')
ax.legend()
ax.set_title('Stock volume: ' + ticker, y=1.01)
#plt.xticks(rotation=70)
plt.show()
```

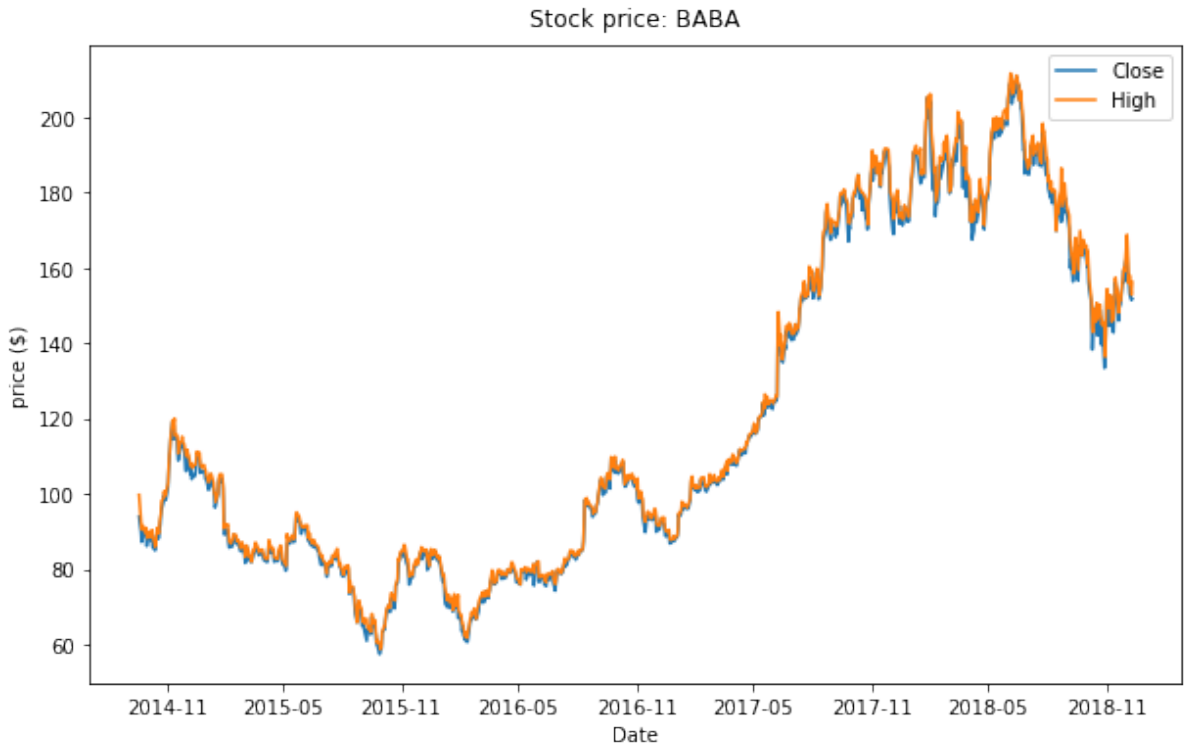


Figure 16.1: Historical Stock Price

16.2.2 Calulate the Compound Annual Growth Rate

The formula for Compound Annual Growth Rate (CAGR) is very useful for investment analysis. It may also be referred to as the annualized rate of return or annual percent yield or effective annual rate, depending on the algebraic form of the equation. Many investments such as stocks have returns that can vary wildly. The CAGR formula allows you to calculate a "smoothed" rate of return that you can use to compare to other investments. The formula is defined as (more details can be found at CAGR Calculator and Formula)

$$\text{CAGR} = \left(\frac{\text{End Value}}{\text{Start Value}}\right)^{\frac{365}{\text{Days}}} - 1$$

```
days =  (stock.Date.iloc[-1] - stock.Date.iloc[0]).days
cagr = ((((stock['Adj Close'].iloc[-1]) / stock['Adj Close'].iloc[0])) ** (365.0/days))
```

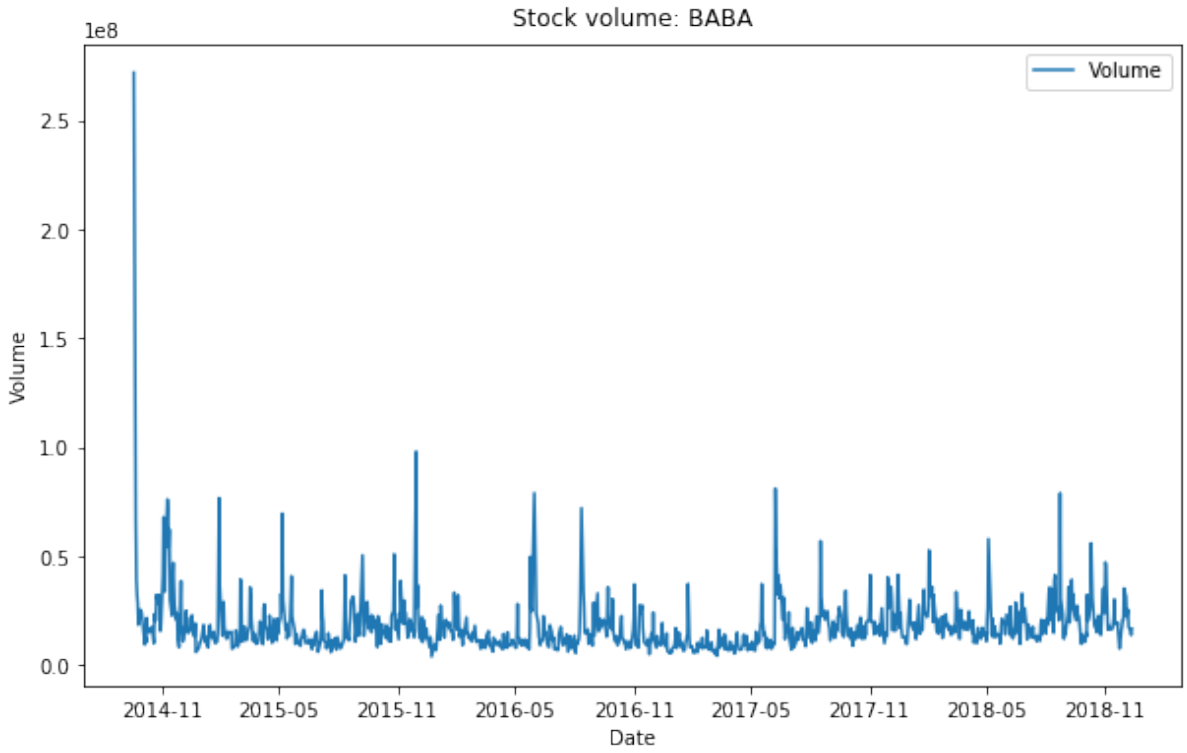


Figure 16.2: Historical Stock Volume

```
print ('CAGR =',str(round(cagr,4)*100)+"%")
mu = cagr
```

16.2.3 Calulate the annual volatility

A stock's volatility is the variation in its price over a period of time. For example, one stock may have a tendency to swing wildly higher and lower, while another stock may move in much steadier, less turbulent way. Both stocks may end up at the same price at the end of day, but their path to that point can vary wildly. First, we create a series of percentage returns and calculate the annual volatility of returns Annualizing volatility. To present this volatility in annualized terms, we simply need to multiply our daily standard deviation by the square root of 252. This assumes there are 252 trading days in a given year. More details can be found at How to Calculate Annualized Volatility.

```
stock['Returns'] = stock['Adj Close'].pct_change()
vol = stock['Returns'].std()*np.sqrt(252)
```

16.2.4 Create matrix of daily returns

1. Create matrix of daily returns using random normal distribution Generates an RDD matrix comprised of i.i.d. samples from the uniform distribution U(0.0, 1.0).

```
S = stock['Adj Close'].iloc[-1] #starting stock price (i.e. last available real stock pr
T = 5 #Number of trading days
mu = cagr #Return
vol = vol #Volatility
trials = 10000
```

```
mat = RandomRDDs.normalVectorRDD(sc, trials, T, seed=1)
```

2. Transform the distribution in the generated RDD from U(0.0, 1.0) to U(a, b), use RandomRDDs.uniformRDD(sc, n, p, seed) .map(lambda v: a + (b - a) * v)

```
a = mu/T
b = vol/math.sqrt(T)
v = mat.map(lambda x: a +  (b - a)* x)
```

3. Convert Rdd mstrix to dataframe

```
df = v.map(lambda x: [round(i,6)+1 for i in x]).toDF()
df.show(5)
```

```
+--------+--------+--------+--------+--------+
|      _1|      _2|      _3|      _4|      _5|
+--------+--------+--------+--------+--------+
|0.935234|1.162894| 1.07972|1.238257|1.066136|
|0.878456|1.045922|0.990071|1.045552|0.854516|
|1.186472|0.944777|0.742247|0.940023|1.220934|
|0.872928|1.030882|1.248644|1.114262|1.063762|
| 1.09742|1.188537|1.137283|1.162548|1.024612|
+--------+--------+--------+--------+--------+
only showing top 5 rows
```

```
from pyspark.sql.functions import lit
S = stock['Adj Close'].iloc[-1]
price = df.withColumn('init_price' ,lit(S))

price.show(5)
```

```
+--------+--------+--------+--------+--------+----------+
|      _1|      _2|      _3|      _4|      _5|init_price|
+--------+--------+--------+--------+--------+----------+
|0.935234|1.162894| 1.07972|1.238257|1.066136|     151.5|
|0.878456|1.045922|0.990071|1.045552|0.854516|     151.5|
|1.186472|0.944777|0.742247|0.940023|1.220934|     151.5|
|0.872928|1.030882|1.248644|1.114262|1.063762|     151.5|
| 1.09742|1.188537|1.137283|1.162548|1.024612|     151.5|
+--------+--------+--------+--------+--------+----------+
only showing top 5 rows
```

```
price = price.withColumn('day_0', col('init_price'))
price.show(5)
```

```
+--------+--------+--------+--------+--------+----------+-----+
|      _1|      _2|      _3|      _4|      _5|init_price|day_0|
+--------+--------+--------+--------+--------+----------+-----+
|0.935234|1.162894| 1.07972|1.238257|1.066136|     151.5|151.5|
|0.878456|1.045922|0.990071|1.045552|0.854516|     151.5|151.5|
|1.186472|0.944777|0.742247|0.940023|1.220934|     151.5|151.5|
|0.872928|1.030882|1.248644|1.114262|1.063762|     151.5|151.5|
| 1.09742|1.188537|1.137283|1.162548|1.024612|     151.5|151.5|
+--------+--------+--------+--------+--------+----------+-----+
only showing top 5 rows
```

16.2.5 Monte Carlo Simulation

```
from pyspark.sql.functions import round
for name in price.columns[:-2]:
    price = price.withColumn('day'+name, round(col(name)*col('init_price'),2))
    price = price.withColumn('init_price',col('day'+name))

price.show(5)
```

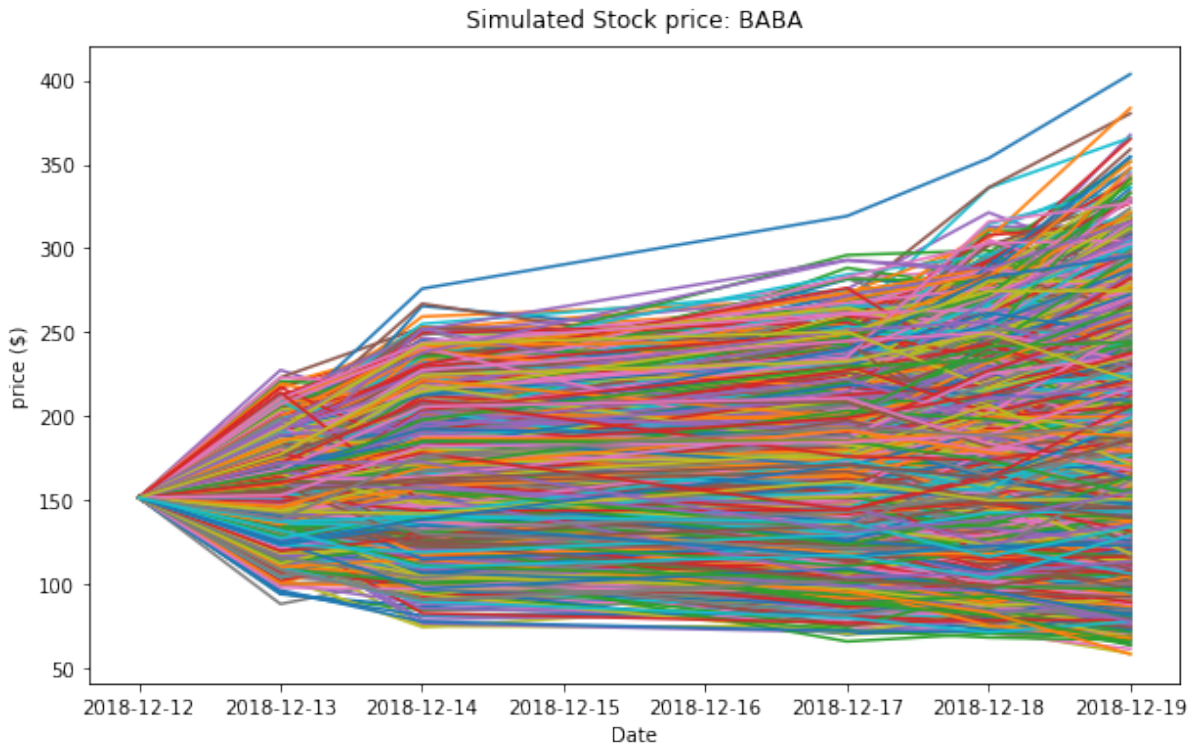
```
+--------+--------+--------+--------+--------+----------+-----+------+------+------+----
|      _1|      _2|      _3|      _4|      _5|init_price|day_0| day_1| day_2| day_3| day
+--------+--------+--------+--------+--------+----------+-----+------+------+------+----
|0.935234|1.162894| 1.07972|1.238257|1.066136|    234.87|151.5|141.69|164.77|177.91| 220
|0.878456|1.045922|0.990071|1.045552|0.854516|    123.14|151.5|133.09| 139.2|137.82| 144
|1.186472|0.944777|0.742247|0.940023|1.220934|    144.67|151.5|179.75|169.82|126.05|118.
|0.872928|1.030882|1.248644|1.114262|1.063762|    201.77|151.5|132.25|136.33|170.23|189.
| 1.09742|1.188537|1.137283|1.162548|1.024612|     267.7|151.5|166.26|197.61|224.74|261.
+--------+--------+--------+--------+--------+----------+-----+------+------+------+----
only showing top 5 rows
```

16.2.6 Summary

```
selected_col = [name for name in price.columns if 'day_' in name]

simulated = price.select(selected_col)
simulated.describe().show()
```

```
+-------+----------+------------------+------------------+------------------+-----------
|summary|2018-12-12|       2018-12-13|       2018-12-14|       2018-12-17|        201
+-------+----------+------------------+------------------+------------------+-----------
|  count|   10000.0|           10000.0|           10000.0|           10000.0|
|   mean|     151.5|155.11643700000002|       158.489058|162.23713200000003|        166
|    std|       0.0|18.313783237787845|26.460919262517276| 33.37780495150803|39.36910107
|    min|     151.5|              88.2|             74.54|             65.87|
|    25%|     151.5|           142.485|            140.15|            138.72|
|    50%|     151.5|            154.97|           157.175|            159.82|
|    75%|     151.5|           167.445|175.48499999999999|          182.8625|
|    max|     151.5|            227.48|            275.94|            319.17|
+-------+----------+------------------+------------------+------------------+-----------
```

```
data_plt = simulated.toPandas()
days = pd.date_range(stock['Date'].iloc[-1], periods= T+1,freq='B').date

width = 10
height = 6
fig = plt.figure(figsize=(width, height))
ax = fig.add_subplot(1,1,1)

days = pd.date_range(stock['Date'].iloc[-1], periods= T+1,freq='B').date

for i in range(trials):
    plt.plot(days, data_plt.iloc[i])
ax.set_xlabel('Date')
ax.set_ylabel('price ($)')
ax.set_title('Simulated Stock price: ' + ticker, y=1.01)
plt.show()
```



16.2.7 One-year Stock price simulation

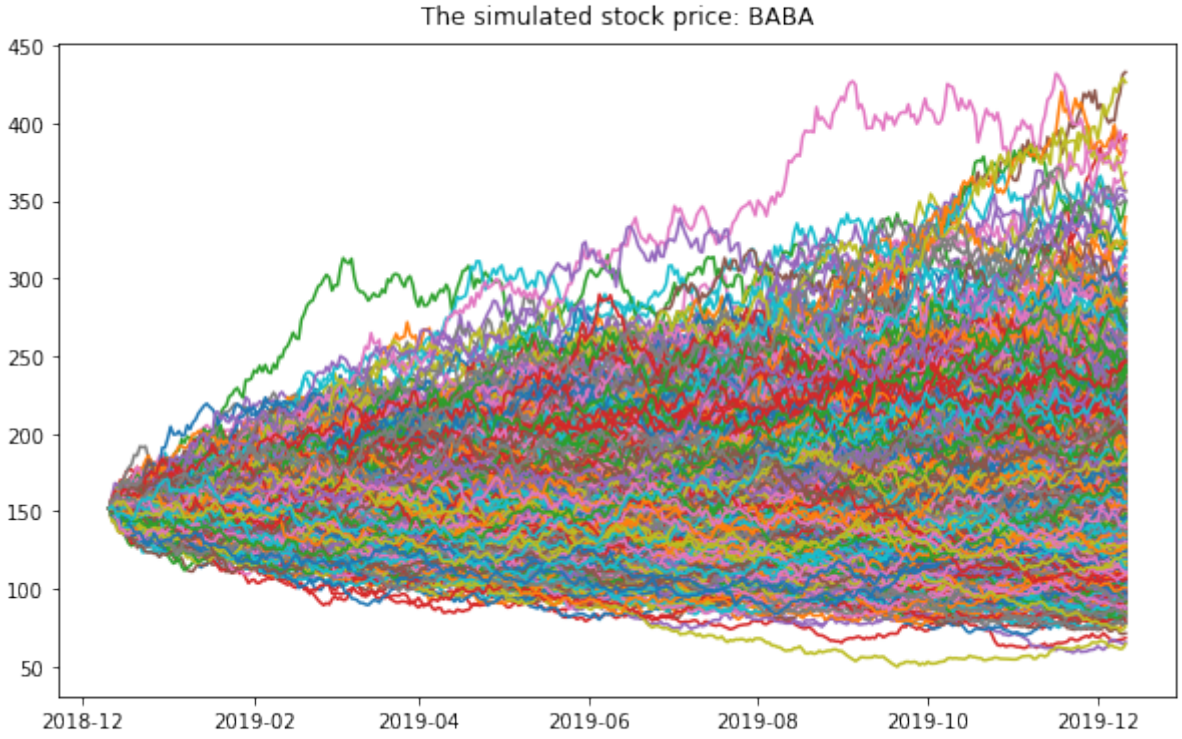


Figure 16.3: Simulated Stock Price

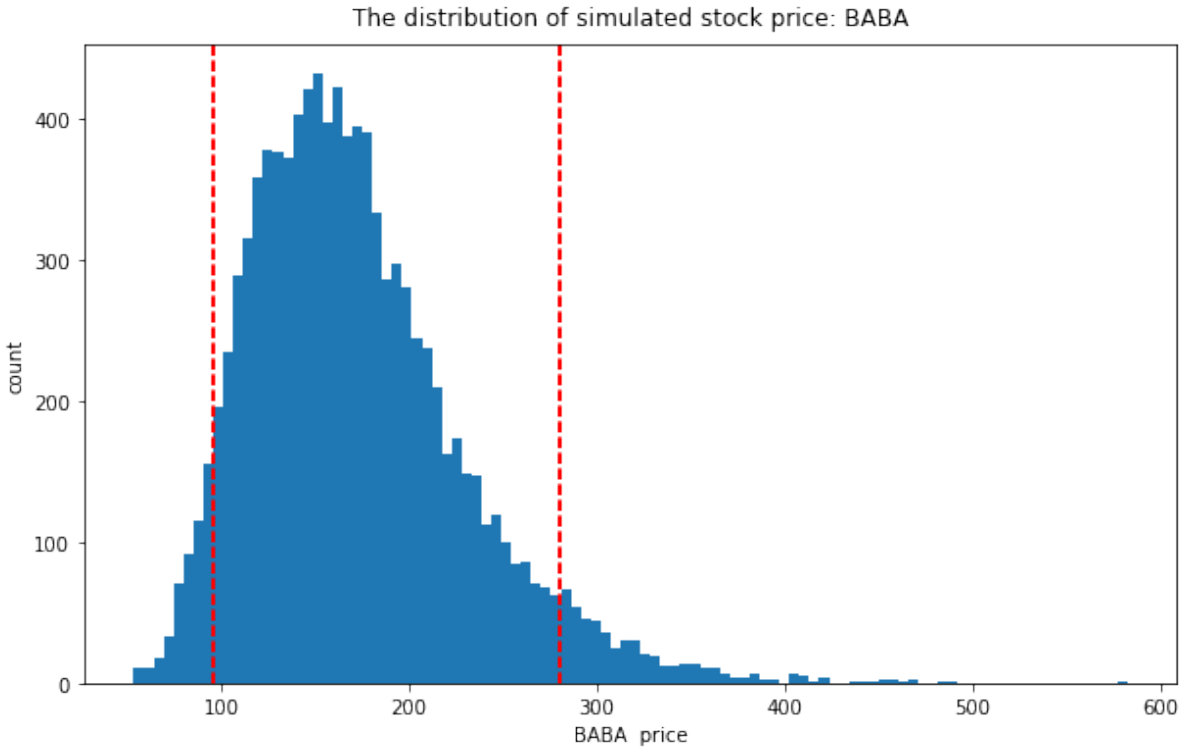


Figure 16.4: Simulated Stock Price distribution

CHAPTER

SEVENTEEN

MARKOV CHAIN MONTE CARLO

Monte Carlo simulations are just a way of estimating a fixed parameter by repeatedly generating random numbers. More details can be found at A Zero Math Introduction to Markov Chain Monte Carlo Methods.

Markov Chain Monte Carlo (MCMC) methods are used to approximate the posterior distribution of a parameter of interest by random sampling in a probabilistic space. More details can be found at A Zero Math Introduction to Markov Chain Monte Carlo Methods.

TODO ..

CHAPTER

EIGHTEEN

NEURAL NETWORK

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb

18.1 Feedforward Neural Network

18.1.1 Introduction

A feedforward neural network is an artificial neural network wherein connections between the units do not form a cycle. As such, it is different from recurrent neural networks.

The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward (see Fig. *MultiLayer Neural Network*), from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

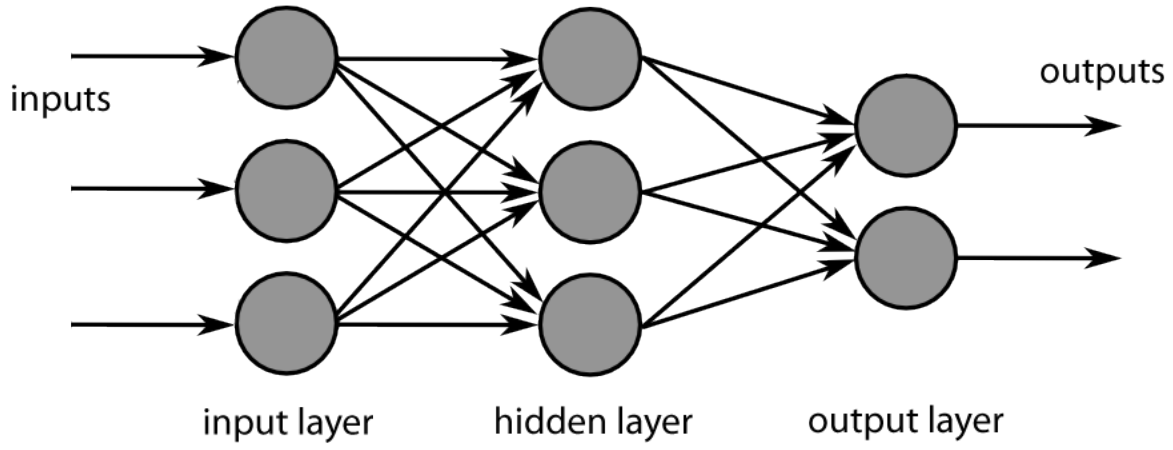


Figure 18.1: MultiLayer Neural Network

18.1.2 Demo

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark Feedforward neural network example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|fixed|volatile|citric|sugar|chlorides|free|total|density|  pH|sulphates|alcohol|quality
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4|      5
|  7.8|    0.88|   0.0|  2.6|    0.098|25.0| 67.0| 0.9968| 3.2|     0.68|    9.8|      5
|  7.8|    0.76|  0.04|  2.3|    0.092|15.0| 54.0|  0.997|3.26|     0.65|    9.8|      5
| 11.2|    0.28|  0.56|  1.9|    0.075|17.0| 60.0|  0.998|3.16|     0.58|    9.8|      6
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|    9.4|      5
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-------+-------
only showing top 5 rows
```

3. change categorical variable size

```
# Convert to float format
def string_to_float(x):
    return float(x)

#
def condition(r):
    if (0<= r <= 4):
        label = "low"
    elif(4< r <= 6):
        label = "medium"
    else:
        label = "high"
    return label

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())
df= df.withColumn("quality", quality_udf("quality"))
```

4. Convert the data to dense vector

```
# convert the data to dense vector
def transData(data):
    return data.rdd.map(lambda r: [r[-1], Vectors.dense(r[:-1])]).\
           toDF(['label','features'])

from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

data= transData(df)
data.show()
```

5. Split the data into training and test sets (40% held out for testing)

```
# Split the data into train and test
(trainingData, testData) = data.randomSplit([0.6, 0.4])
```

6. Train neural network

```
# specify layers for the neural network:
# input layer of size 11 (features), two intermediate of size 5 and 4
# and output of size 7 (classes)
layers = [11, 5, 4, 4, 3 , 7]

# create the trainer and set its parameters
FNN = MultilayerPerceptronClassifier(labelCol="indexedLabel", \
                                     featuresCol="indexedFeatures",\
                                     maxIter=100, layers=layers, \
                                     blockSize=128, seed=1234)
# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)
# Chain indexers and forest in a Pipeline
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, FNN, labelConverter])
# train the model
# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

7. Make predictions

```
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

8. Evaluation

```
# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Predictions accuracy = %g, Test Error = %g" % (accuracy,(1.0 - accuracy)))
```

CHAPTER

NINETEEN

MY PYSPARK PACKAGE

It's super easy to wrap your own package in Python. I packed some functions which I frequently used in my daily work. You can download and install it from My PySpark Package. The hierarchical structure and the directory structure of this package are as follows.

19.1 Hierarchical Structure

```
|-- build
|   |-- bdist.linux-x86_64
|   |-- lib.linux-x86_64-2.7
|       |-- PySparkTools
|           |-- __init__.py
|           |-- Manipulation
|           |   |-- DataManipulation.py
|           |   |-- __init__.py
|           |-- Visualization
|               |-- __init__.py
|               |-- PyPlots.py
|-- dist
|   |-- PySParkTools-1.0-py2.7.egg
|-- __init__.py
|-- PySparkTools
|   |-- __init__.py
|   |-- Manipulation
|   |   |-- DataManipulation.py
|   |   |-- __init__.py
|   |-- Visualization
|       |-- __init__.py
|       |-- PyPlots.py
|       |-- PyPlots.pyc
|-- PySParkTools.egg-info
|   |-- dependency_links.txt
|   |-- PKG-INFO
|   |-- requires.txt
|   |-- SOURCES.txt
|   |-- top_level.txt
|-- README.md
|-- requirements.txt
|-- setup.py
|-- test
    |-- spark-warehouse
    |-- test1.py
    |-- test2.py
```

From the above hierarchical structure, you will find that you have to have `__init__.py` in each directory. I will explain the `__init__.py` file with the example below:

19.2 Set Up

```
from setuptools import setup, find_packages

try:
    with open("README.md") as f:
        long_description = f.read()
except IOError:
    long_description = ""

try:
    with open("requirements.txt") as f:
        requirements = [x.strip() for x in f.read().splitlines() if x.strip()]
except IOError:
    requirements = []

setup(name='PySParkTools',
          install_requires=requirements,
      version='1.0',
      description='Python Spark Tools',
      author='Wenqiang Feng',
      author_email='von198@gmail.com',
      url='https://github.com/runawayhorse001/PySparkTools',
      packages=find_packages(),
      long_description=long_description
     )
```

19.3 ReadMe

```
# PySparkTools

This is my PySpark Tools. If you want to colne and install it, you can use

- clone

'''{bash}
git clone git@github.com:runawayhorse001/PySparkTools.git
'''
- install

'''{bash}
cd PySparkTools
pip install -r requirements.txt
python setup.py install
'''

- test

'''{bash}
cd PySparkTools/test
```

```
python test1.py
```
```
```

CHAPTER

TWENTY

MAIN REFERENCE

[Bird2009] 19. Bird, E. Klein, and E. Loper. Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc., 2009.

[Feng2017] 23. Feng and M. Chen. Learning Apache Spark, Github 2017.

[Karau2015] 8. Karau, A. Konwinski, P. Wendell and M. Zaharia. Learning Spark: Lightning-Fast Big Data Analysis. O'Reilly Media, Inc., 2015

[Kirillov2016] Anton Kirillov. Apache Spark: core concepts, architecture and internals. http://datastrophic.io/core-concepts-architecture-and-internals-of-apache-spark/

C

R

S