电子科技大学信息与软件工程学院

**项 目 报 告**

课程名称 大数据分析与智能计算

理论教师 汤羽

实验教师 李美蓉

学生信息：

|  |
| --- |
| 序号 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**项 目 报 告**

**指导教师：李美蓉 地点：二教107**

1. **项目名称：航空公司****延误和取消分析项目**
2. **项目时间：2022.12.2—2022.1.2**
3. **项目原理**

读入航班数据并处理（可以存入HDFS、HBase等），制作图形和表格来进行数据分析，建立预测模型，预测航班取消情况，分析至少包括以下内容：①查看飞机延误时间最长的前10名航班。②计算延误的和没有延误的航空公司的比例。③分析一天中、一周中延误最严重的飞行时间。④短途航班和长途航班，哪种航班取消更严重？⑤建立机器学习算法模型，预测未来航班取消情况。

针对DataExpo2009数据集，通过Python、PySpark等工具，进行航空公司延误和取消分析。该数据集包含1987年10月到2008年12月美国境内所有商业航班的航班到达和离开详细信息。这是一个大型数据集：总共有近1.2亿条记录，占用了1.6 GB的压缩空间和12 GB的未压缩时空间。

Python是数据分析最常用的语言之一，而Apache Spark是一个开源的强大的分布式查询和处理引擎。本实验要求基于Python语言进行Spark Application编程，完成数据获取、处理、数据分析及可视化方面常用的数据分析方法与技巧，让学生掌握使用PySpark来分析数据。

预测模型选择

模型可选择：支持向量机（SVM）、随机森林（Random Forest）、梯度提升树(GBDT)、线性判别分析（Linear Discriminant Analysis）、伯努利贝叶斯分类（BernoulliNB）、Adaboost、XGBoost等。

模型评估方法可用：准确率和召回率、ROC 曲线、AUC（ROC 曲线下的面积）等，按照评分高低进行比较。如果要用深度学习来训练算法模型，由于数据集涉及到时间序列相关，建议选择长短期记忆模型（LSTM）。

# 项目内容

1. PySpark的安装及测试
2. Jupyter Notebook安装及测试
3. 针对数据集，进行相应的数据分析
   1. 查看飞机延误时间最长的前10名航班。
   2. 计算延误的和没有延误的航空公司的比例。
   3. 分析一天中延误最严重的飞行时间
   4. 分析一周中延误最严重的飞行时间。
   5. 短途航班和长途航班，哪种航班取消更严重？
   6. 建立机器学习算法模型，预测未来航班取消情况。

# 五、 需求分析与设计

1. 背景

近年以来，受各种因素的影响，航班的正常率一直维持在一个比较低的水平，航班延误已经成为制约航空发展的一个短板。同时航班作为公共交通运输服务的一个重要成员，其为社会公共服务的质量，一定程度上损害了公众利益。尤其是出现航班大面积延误时，旅客与航空运输企业的纠纷就越来越多，矛盾越来越突出，严重时甚至激化为冲突。

1. 任务概述
   1. 目标

本项目的任务是读入航班数据并处理（可以存入HDFS、HBase等），制作图形和表格来进行数据分析，建立预测模型，并预测航班取消情况本实验要求基于Python语言进行Spark Application编程，完成数据获取、处理、数据分析及可视化方面常用的数据分析方法与技巧，聚焦于一个基础的数据集，该数据集包含1987年10月到2008年12月美国境内所有商业航班的航班到达和离开详细信息，主要解决五大类问题：查看飞机延误时间最长的前10名航班、计算延误的和没有延误的航空公司的比例、分析一段时间中延误最严重的飞行时间、分析短途航班和长途航班哪种航班取消更严重以及对于将来航班取消情况做出预测，层层深入，为用户切实解决航班延误问题。

本项目开发的目标是，帮助航班使用者更便捷获取航班延误情况资讯，提供一个集查询、分析、预测于一体的平台。

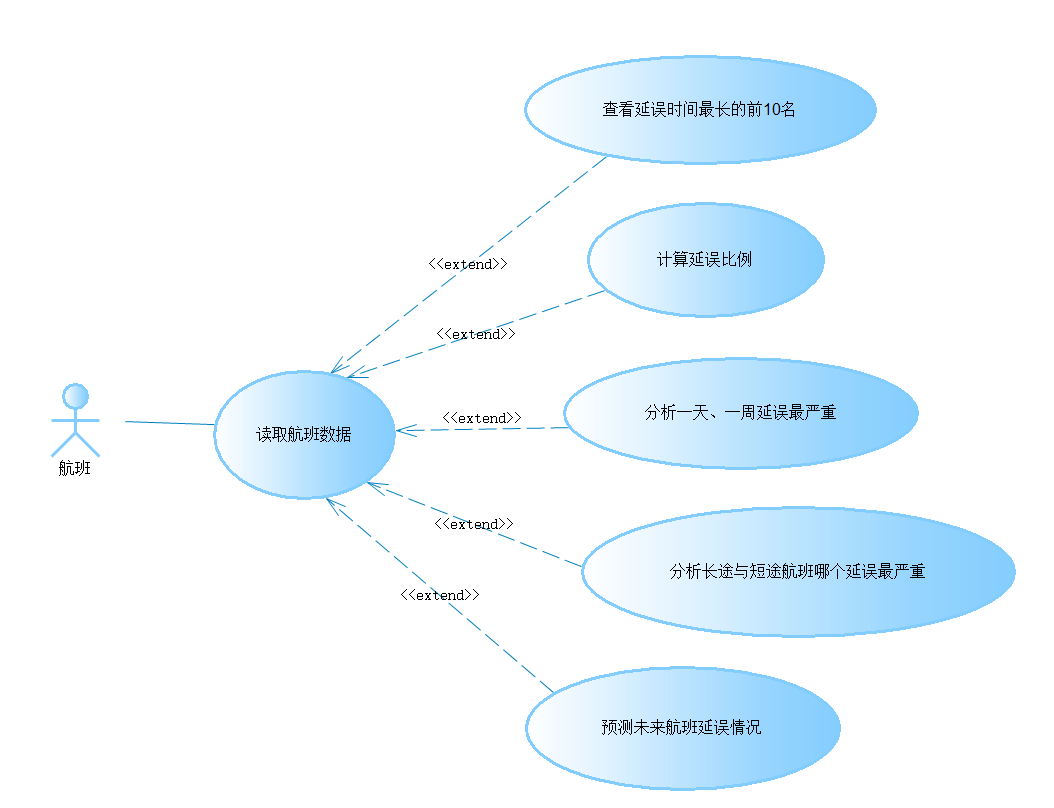
* 1. 用户特点

用户特点：范围较广，暂无客观规律性

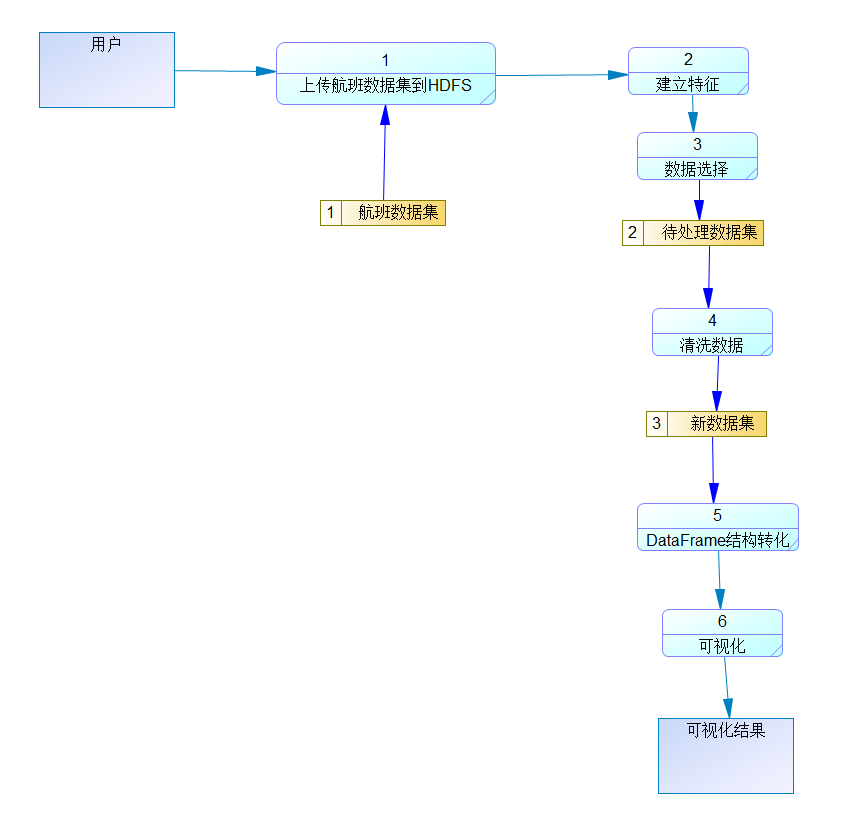
1. 运行环境规定

本项目是基于VMware虚拟机的ubuntu-20.04.3系统，采用Anaconda开发工具完成。

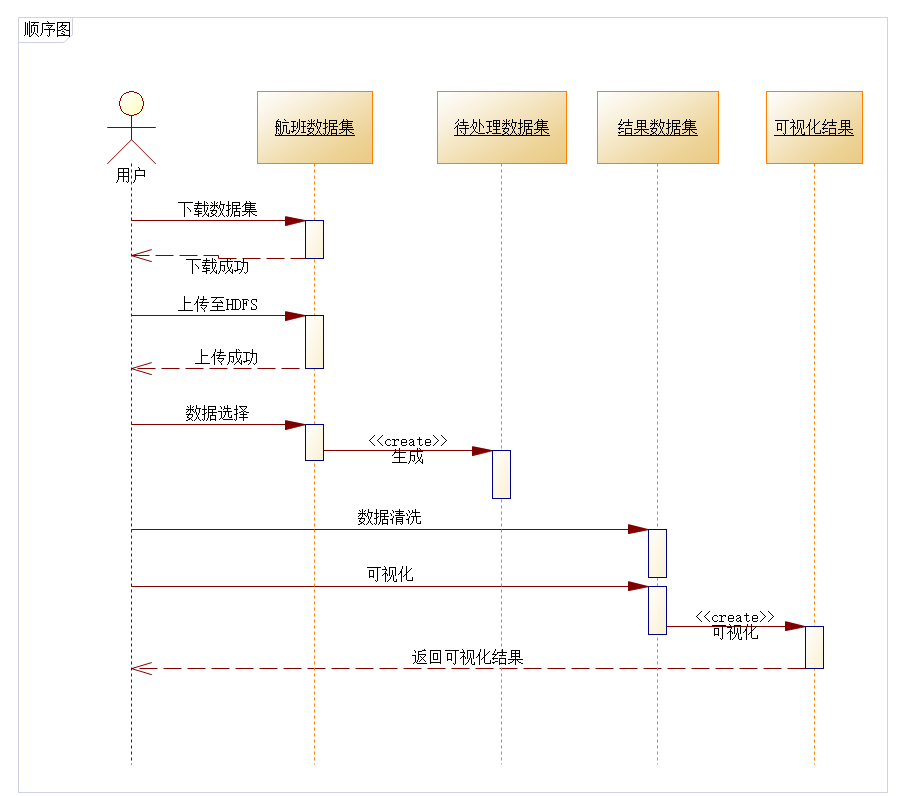
1. 用例图



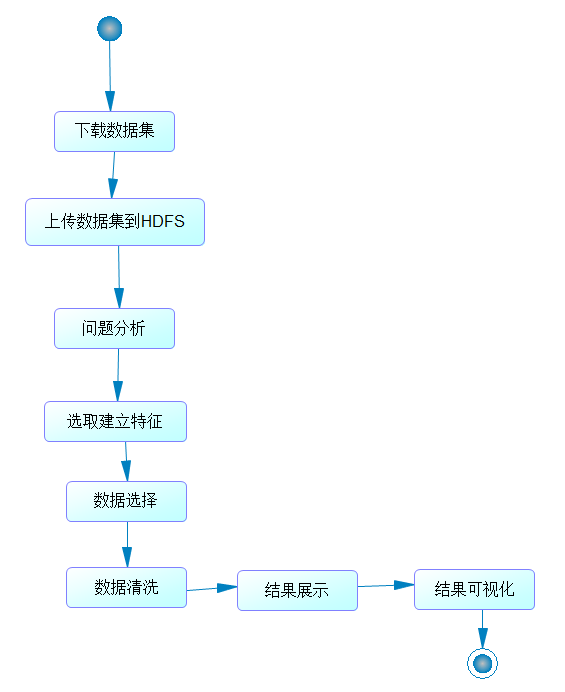
1. 数据流图



1. 顺序图



1. 流程图



# 项目计划

1. 进行需求分析，绘制用例图等详细描述需求分析
2. 环境配置，安装PySpark、Jupyter Notebook
3. 测试实验环境
4. 准备实验数据集
5. 针对数据集，进行相应的数据分析

# 七、项目环境配置管理

**7.1** **操作系统**：VMware虚拟机ubuntu-20.04.3系统针对数据集，进行相应的数据分析

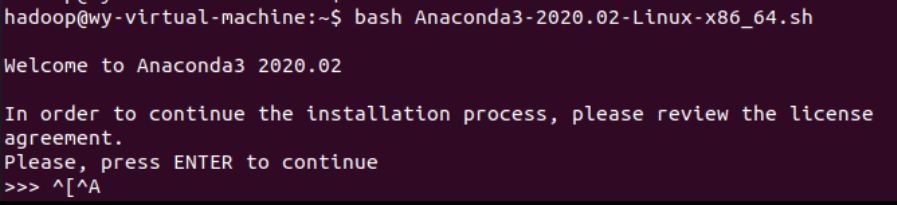
**7.2 开发工具**：Anaconda

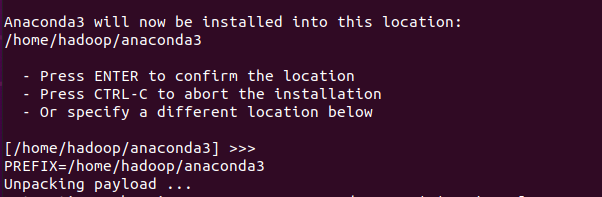
**7.3 配置过程**

1. **PySpark安装配置**
2. 下载Anaconda包

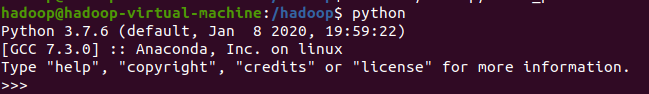


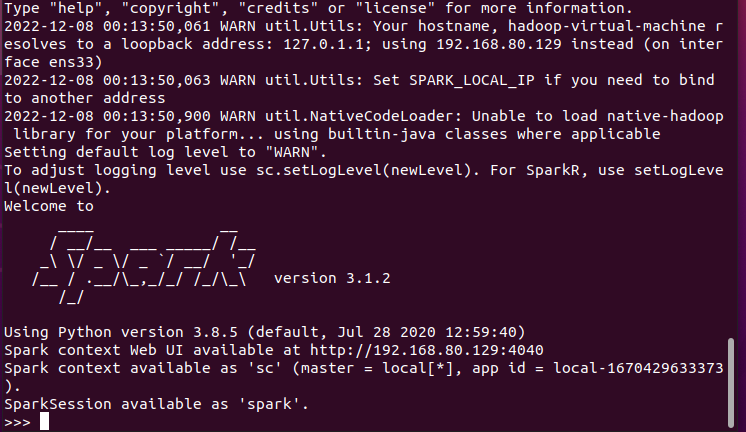
1. 安装Anaconda



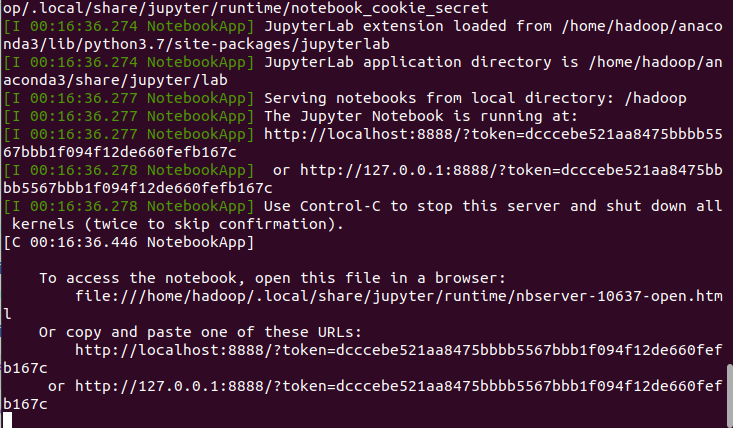


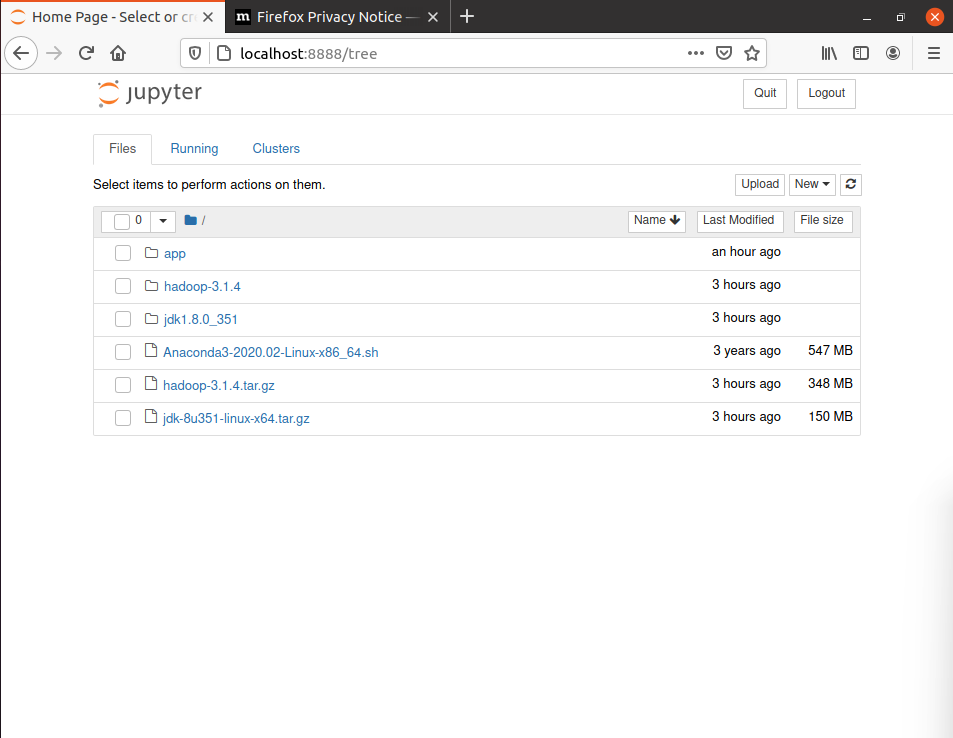
1. 验证python环境





1. **Jupyter安装配置**
   1. 配置PySpark driver
   2. 启动Jupyter





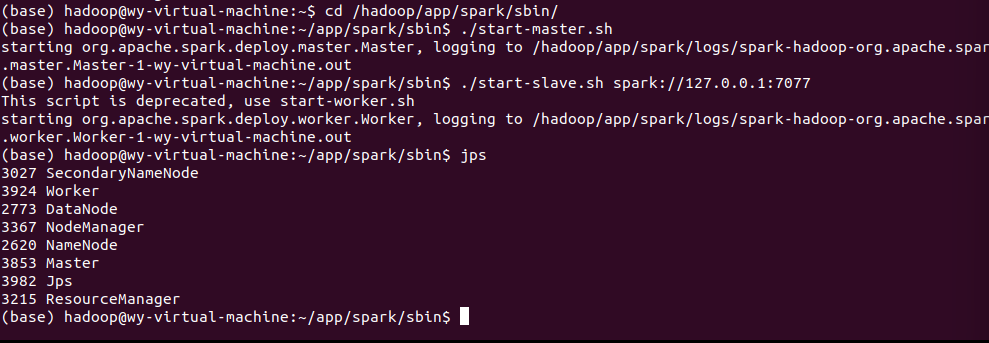
# 八、项目实践过程

1. **准备过程**
2. 启动Hadoop



上图显示启动成功。

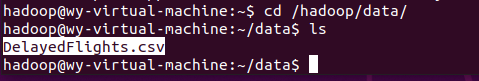
1. 启动Spark



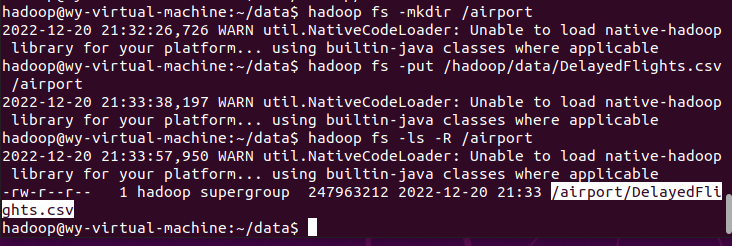
上图显示启动成功。

1. 上传文件

将数据集文件导入到hadoop用户中：



将数据集文件上传到hdfs系统中：



1. **问题一**

（1） 问题分析

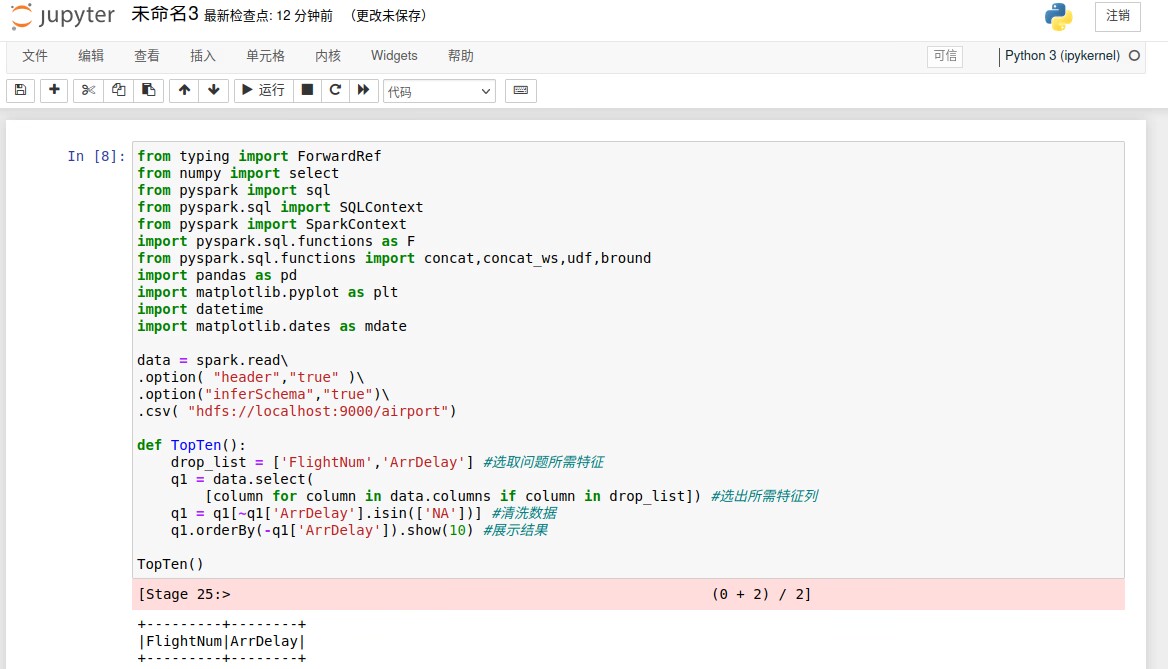
问题一需要“查看飞机延误时间最长的前10名航班”，经过讨论和分析，我们认为该问题应该按照抵达延误时间对航班进行排序，所以需要提取数据集中特征**航班号FlightNum、抵达延误时间ArrDelay，按照ArrDelay**大小进行排序。

（2）代码

代码 1

|  |
| --- |
| from typing import ForwardRef  from numpy import select  from pyspark import sql  from pyspark.sql import SQLContext  from pyspark import SparkContext  import pyspark.sql.functions as F  from pyspark.sql.functions import concat,concat\_ws,bround  import pandas as pd  import matplotlib.pyplot as plt  import datetime  import matplotlib.dates as mdate  data = spark.read\  .option("header","true")\  .option("inferSchema","true")\  .csv("hdfs://localhost:9000/airport")  def TopTen():  drop\_list = ['FlightNum','ArrDelay'] #选取问题所需特征  q1 = data.select(  [column for column in data.columns if column in drop\_list]) #选出所需特征列  q1 = q1[~q1['ArrDelay'].isin(['NA'])] #清洗数据  q1.orderBy(-q1['ArrDelay']).show(10) #展示结果  TopTen() |

运行页面如下：



1. **问题二**
2. 问题分析

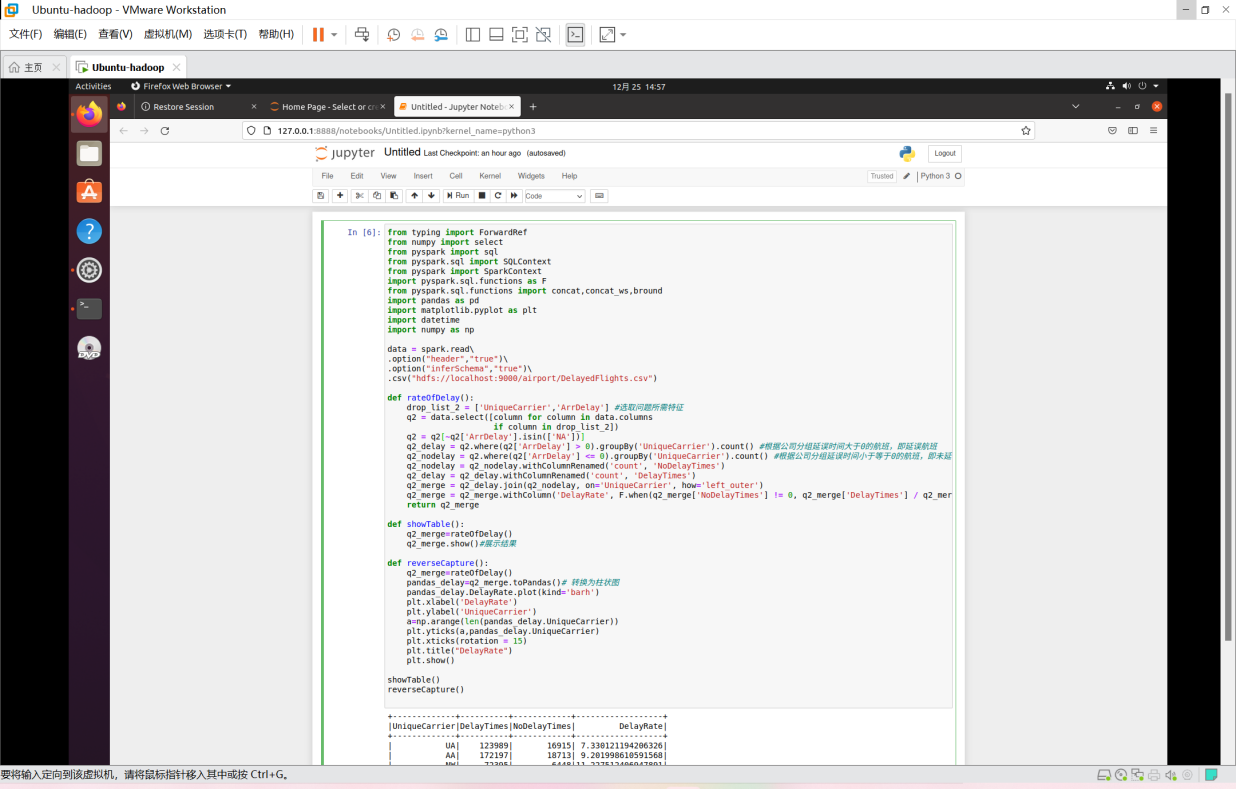
本问题为“计算延误的和没有延误的航空公司的比例”，在此问题中，建立特征为**航空公司编码UniqueCarrier**和**抵达的延误时间ArrDelay**，ArrDelay的作用是：根据判断ArrDelay是否大于零判断是否延误，继而分别进行统计、计算比例。

1. 代码

代码 2

|  |
| --- |
| from typing import ForwardRef  from numpy import select  from pyspark import sql  from pyspark.sql import SQLContext  from pyspark import SparkContext  import pyspark.sql.functions as F  from pyspark.sql.functions import concat,concat\_ws,bround  import pandas as pd  import matplotlib.pyplot as plt  import datetime  import numpy as np  data = spark.read\  .option("header","true")\  .option("inferSchema","true")\  .csv("hdfs://localhost:9000/airport/DelayedFlights.csv")  def rateOfDelay():  #建立特征  drop\_list\_2 = ['UniqueCarrier','ArrDelay']  q2 = data.select([column for column in data.columns  if column in drop\_list\_2])  q2 = q2[~q2['ArrDelay'].isin(['NA'])]  q2\_delay = q2.where(q2['ArrDelay'] > 0).groupBy('UniqueCarrier').count() #将延误时间大于零的航班分组，生成延误航班组  q2\_nodelay = q2.where(q2['ArrDelay'] <= 0).groupBy('UniqueCarrier').count() #将延误时间小于等于0的航班重新分组 ，生成未延误航班组  q2\_nodelay = q2\_nodelay.withColumnRenamed('count', 'NoDelayTimes') #重命名未延误航班列名  q2\_delay = q2\_delay.withColumnRenamed('count', 'DelayTimes') #重命名延误航班列名  q2\_merge = q2\_delay.join(q2\_nodelay, on='UniqueCarrier', how='left\_outer') #将延误航班与未延误航班列合并  q2\_merge = q2\_merge.withColumn('DelayRate', F.when(q2\_merge['NoDelayTimes'] != 0, q2\_merge['DelayTimes'] / q2\_merge['NoDelayTimes']).otherwise(1) )  return q2\_merge #返回新生成的数据集  def showTable():  q2\_merge=rateOfDelay()  q2\_merge.show()#展示结果    def reverseCapture():  q2\_merge=rateOfDelay()  pandas\_delay=q2\_merge.toPandas()# 转换为柱状图  pandas\_delay.DelayRate.plot(kind='barh')  plt.xlabel('DelayRate')  plt.ylabel('UniqueCarrier')  a=np.arange(len(pandas\_delay.UniqueCarrier))  plt.yticks(a,pandas\_delay.UniqueCarrier)#设置y轴  plt.xticks(rotation = 15)#设置x轴  plt.title("DelayRate")  plt.show()#展示柱状图  showTable()  reverseCapture() |

运行页面如下：



1. **问题三**

（1）问题分析

问题三要求我们分别“分析一天中、一周中延误最严重的飞行时间”，首先经过分析，我们理解题意为需要得到的数据结果为数据集内每一天、每一周中的最高延误时间，所以我们选取特征为年Year、月Month、月中的某日DayofMonth、星期几DayOfWeek、抵达延误时间ArrDelay，先算出每一天的最高延误时间，再据此算出每一周的最高延误时间。

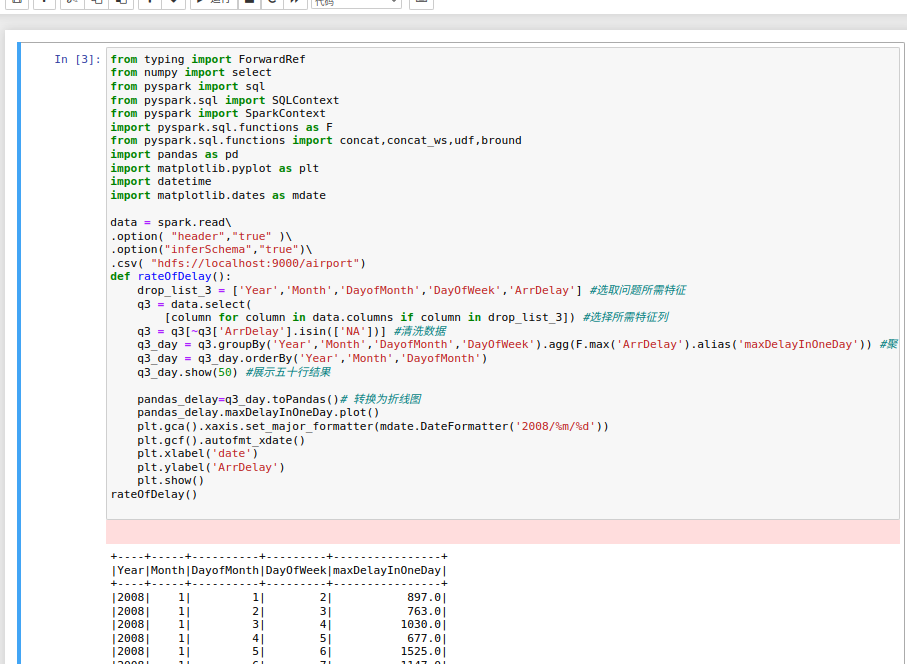
（2）代码

A.每一天的最高延误时间(只显示前50行)

代码 3

|  |
| --- |
| from typing import ForwardRef  from numpy import select  from pyspark import sql  from pyspark.sql import SQLContext  from pyspark import SparkContext  import pyspark.sql.functions as F  from pyspark.sql.functions import concat,concat\_ws,udf,bround  import pandas as pd  import matplotlib.pyplot as plt  import datetime  import matplotlib.dates as mdate  data = spark.read\  .option( "header","true" )\  .option("inferSchema","true")\  .csv( "hdfs://localhost:9000/airport")  def rateOfDelay():  drop\_list\_3 = ['Year','Month','DayofMonth','DayOfWeek','ArrDelay'] #选取问题所需特征  q3 = data.select(  [column for column in data.columns if column in drop\_list\_3]) #选择所需特征列  q3 = q3[~q3['ArrDelay'].isin(['NA'])] #清洗数据  q3\_day = q3.groupBy('Year','Month','DayofMonth','DayOfWeek').agg(F.max('ArrDelay').alias('maxDelayInOneDay')) #聚合每天的抵达延误时间进行比较，选出最大值加入“maxDelayInOneDay”列中  q3\_day = q3\_day.orderBy('Year','Month','DayofMonth')  q3\_day.show(50) #展示五十行结果    pandas\_delay=q3\_day.toPandas()# 转换为折线图  pandas\_delay.maxDelayInOneDay.plot()  plt.gca().xaxis.set\_major\_formatter(mdate.DateFormatter('2008/%m/%d'))  plt.gcf().autofmt\_xdate()  plt.xlabel('date')  plt.ylabel('ArrDelay')  plt.show()  rateOfDelay() |

运行界面如下：

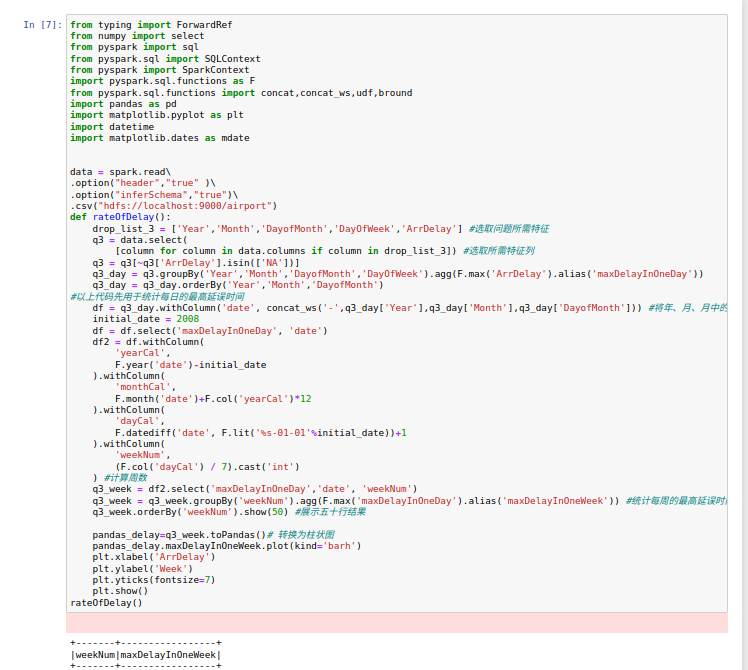


B. 每一周的最高延误时间(只显示前50行)

代码 4

|  |
| --- |
| from typing import ForwardRef  from numpy import select  from pyspark import sql  from pyspark.sql import SQLContext  from pyspark import SparkContext  import pyspark.sql.functions as F  from pyspark.sql.functions import concat,concat\_ws,udf,bround  import pandas as pd  import matplotlib.pyplot as plt  import datetime  import matplotlib.dates as mdate  data = spark.read\  .option("header","true" )\  .option("inferSchema","true")\  .csv("hdfs://localhost:9000/airport")  def rateOfDelay():  drop\_list\_3 = ['Year','Month','DayofMonth','DayOfWeek','ArrDelay'] #选取问题所需特征  q3 = data.select(  [column for column in data.columns if column in drop\_list\_3]) #选取所需特征列  q3 = q3[~q3['ArrDelay'].isin(['NA'])]  q3\_day = q3.groupBy('Year','Month','DayofMonth','DayOfWeek').agg(F.max('ArrDelay').alias('maxDelayInOneDay'))  q3\_day = q3\_day.orderBy('Year','Month','DayofMonth')  #以上代码先用于统计每日的最高延误时间  df = q3\_day.withColumn('date', concat\_ws('-',q3\_day['Year'],q3\_day['Month'],q3\_day['DayofMonth'])) #将年、月、月中的第几日用分隔符“-”连接，与date并列  initial\_date = 2008  df = df.select('maxDelayInOneDay', 'date')  df2 = df.withColumn(  'yearCal',  F.year('date')-initial\_date  ).withColumn(  'monthCal',  F.month('date')+F.col('yearCal')\*12  ).withColumn(  'dayCal',  F.datediff('date', F.lit('%s-01-01'%initial\_date))+1  ).withColumn(  'weekNum',  (F.col('dayCal') / 7).cast('int')  ) #计算周数  q3\_week = df2.select('maxDelayInOneDay','date', 'weekNum')  q3\_week = q3\_week.groupBy('weekNum').agg(F.max('maxDelayInOneDay').alias('maxDelayInOneWeek')) #统计每周的最高延误时间  q3\_week.orderBy('weekNum').show(50) #展示五十行结果    pandas\_delay=q3\_week.toPandas()# 转换为柱状图  pandas\_delay.maxDelayInOneWeek.plot(kind='barh')  plt.xlabel('ArrDelay')  plt.ylabel('Week')  plt.yticks(fontsize=7)  plt.show()  rateOfDelay() |

运行程序界面如下：



1. **问题四**
2. 问题分析

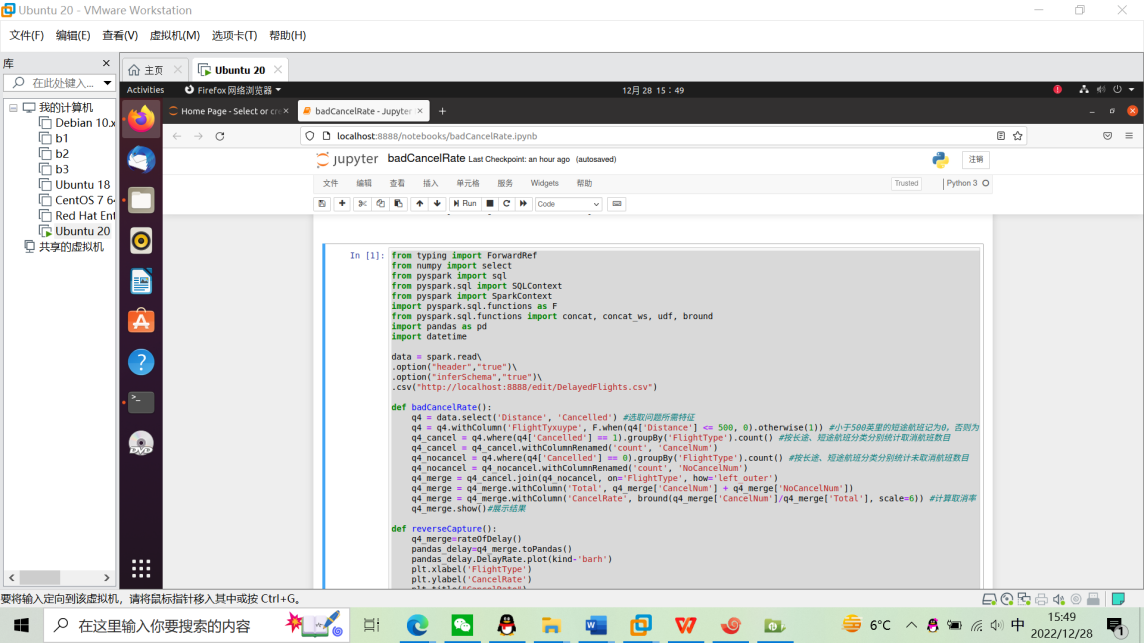
这一问要求判断“短途航班和长途航班，哪种航班取消更严重”，我们对该问题的分析为首先拟定短途和长途航班的距离分割界限为500英里，然后首先对所有航班进行短途和长途的分类，再分别同济两类中的是否取消情况，据此分析，我们需要的特征有**航行距离Distance、航班是否取消Cancelled**。

1. 代码

代码 5

|  |
| --- |
| from typing import ForwardRef  from numpy import select  from pyspark import sql  from pyspark.sql import SQLContext  from pyspark import SparkContext  import pyspark.sql.functions as F  from pyspark.sql.functions import concat, concat\_ws, udf, bround  import pandas as pd  import datetim0e  data = spark.read\  .option("header","true")\  .option("inferSchema","true")\  .csv("http://localhost:8888/edit/DelayedFlights.csv")  def badCancelRate():  q4 = data.select('Distance', 'Cancelled') #选取问题所需特征  q4 = q4.withColumn('FlightTyxuype', F.when(q4['Distance'] <= 500, 0).otherwise(1)) #小于500英里的短途航班记为0，否则为长途航班，记为1  q4\_cancel = q4.where(q4['Cancelled'] == 1).groupBy('FlightType').count() #按长途、短途航班分类分别统计取消航班数目  q4\_cancel = q4\_cancel.withColumnRenamed('count', 'CancelNum')  q4\_nocancel = q4.where(q4['Cancelled'] == 0).groupBy('FlightType').count() #按长途、短途航班分类分别统计未取消航班数目  q4\_nocancel = q4\_nocancel.withColumnRenamed('count', 'NoCancelNum')  q4\_merge = q4\_cancel.join(q4\_nocancel, on='FlightType', how='left\_outer')  q4\_merge = q4\_merge.withColumn('Total', q4\_merge['CancelNum'] + q4\_merge['NoCancelNum'])  q4\_merge = q4\_merge.withColumn('CancelRate', bround(q4\_merge['CancelNum']/q4\_merge['Total'], scale=6)) #计算取消率  q4\_merge.show()#展示结果    def reverseCapture():  q4\_merge=rateOfDelay()  pandas\_delay=q4\_merge.toPandas()  pandas\_delay.DelayRate.plot(kind-'barh')  plt.xlabel('FlightType')  plt.ylabel('CancelRate')  plt.title("CancelRate")  plt.show()# |

1. 运行界面



1. **问题五**

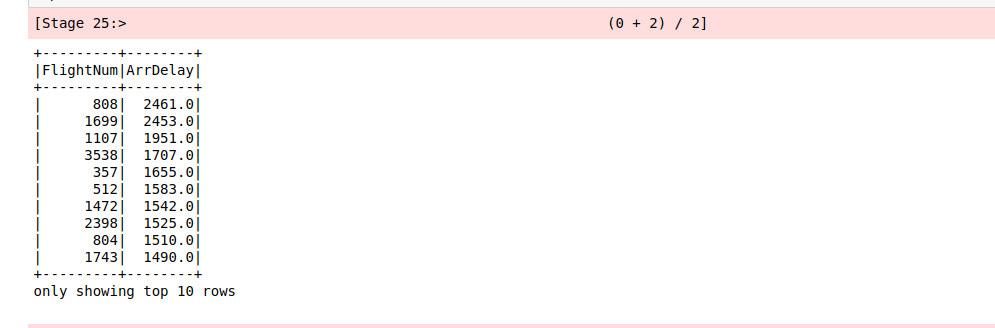
机器学习算法模型以预测未来航班取消情况的代码与运行界面如九-6所示。

**九、项目结果与分析（含重要数据结果分析或核心代码流程分析）**

**1. 查看飞机延误时间最长的前10名航班。**

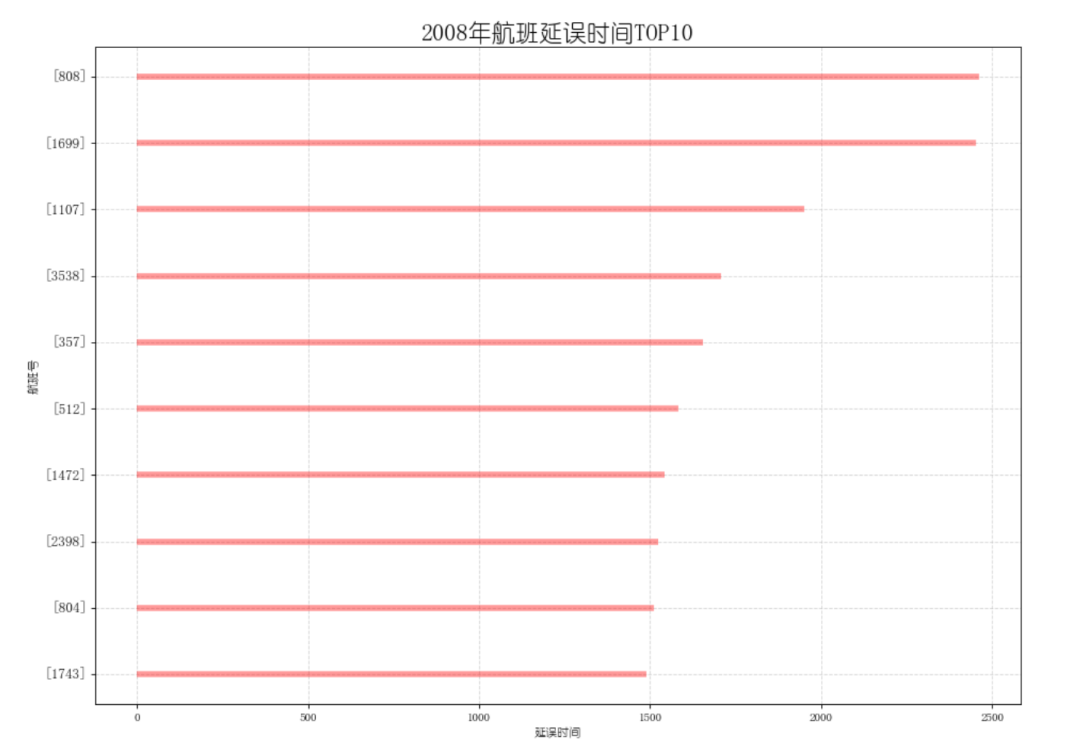
(1)重要数据结果分析

通过运行代码1，得到下图所示的结果。以下列表展示为数据集中前10名延误时间最长的航班，第一列为航班号，第二列为延误时间。



（2）柱状图表示

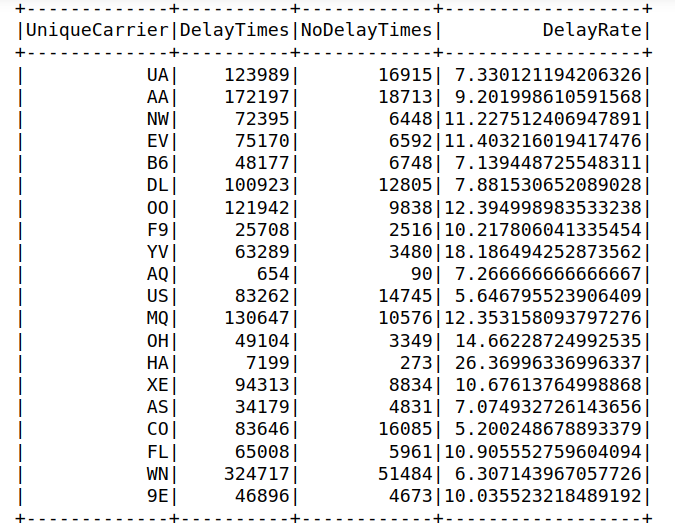
我们将上面的结果转换为可视化表格，将结果从pyspark下的dataframe结构转化为pandas下的dataframe，然后调用对pandas中对dataframe的画图接口plot()进行绘制。纵坐标为航班号，横坐标为延误时间，从上至下，可以一目了然地查看飞机延误时间最长的前10名航班，与上表符合，结果如下：



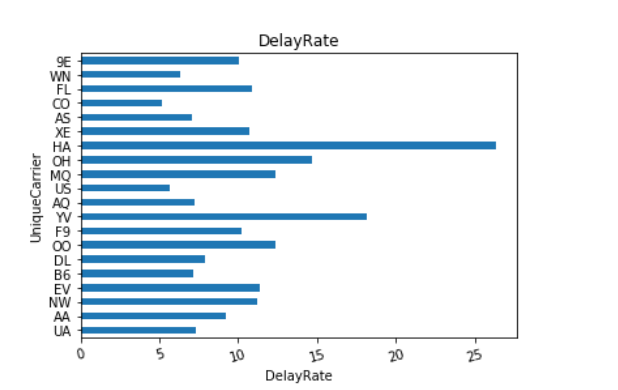
**2. 计算延误的和没有延误的航空公司的比例。**

(1)重要数据结果分析

通过运行代码二，得到下图所示的结果，展示了航空公司中延误与未延误的航空比例；第一列为航空公司编码，第二列为延误航班数，第三列为未延误航班数，最后一列未延误比例。通过结果可以看出HA航空公司延误比例最高，CO最低。



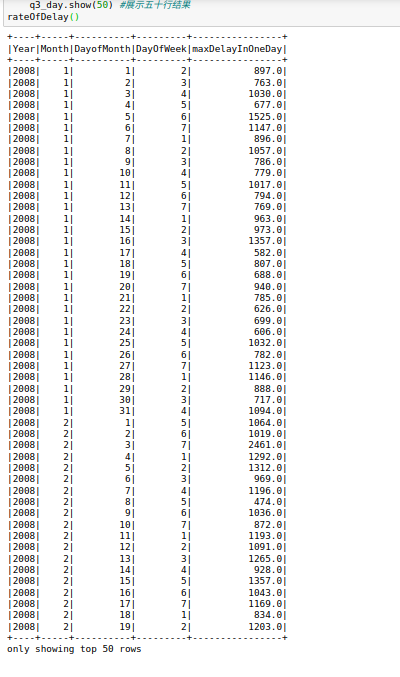
我们将上面的表格转换为柱状图，从柱状图中可以更明显地观察出HA的延误比例最高，CO最低，这个结果与上表相同。



**3. 分析****一天中延误最严重的飞行时间。**

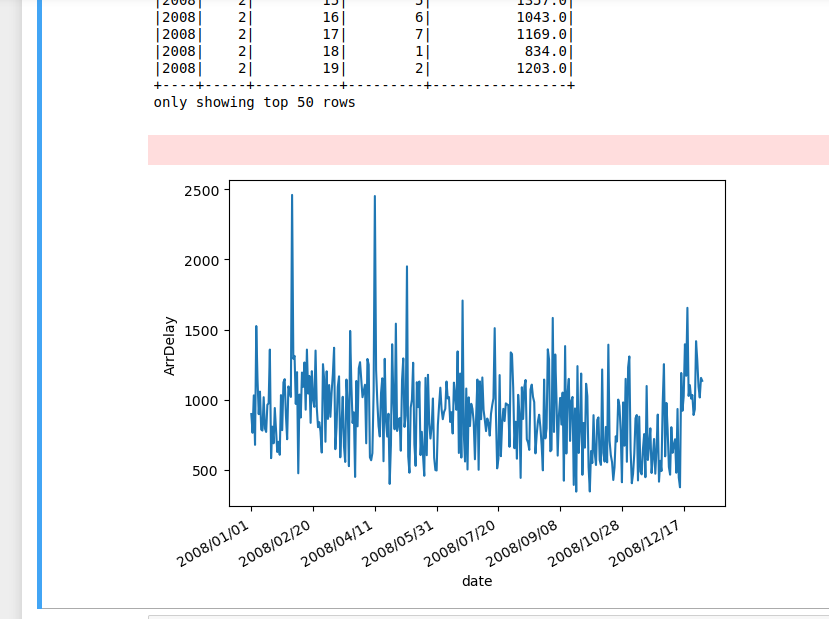
（1）重要数据结果分析

以下列表展示一年中每天最高的抵达延误时间，第一至四列分别为年、月、月中的第几日、星期几，第五列为该天最高的抵达延误时间。由于篇幅有限，此处仅展示部分结果（前50行）。



（2）折线图表示

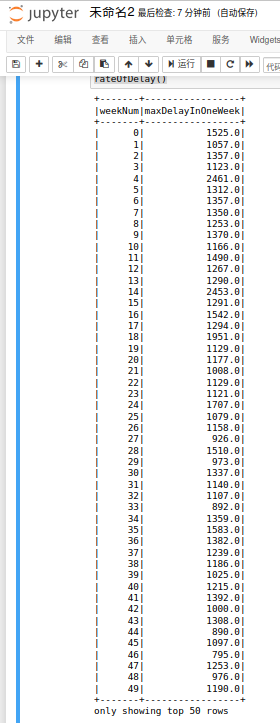
我们将上面的结果转换为可视化表格，调用toPandas()函数将pyspark下的dataframe结构转化为pandas下的dataframe，然后调用plot（）绘制折线图。该折线图用于每日延误的峰值，y轴代表的是延误时间，x轴代表的是日期，图形如下图所示：



**4. 分析一周中延误最严重的飞行时间。**

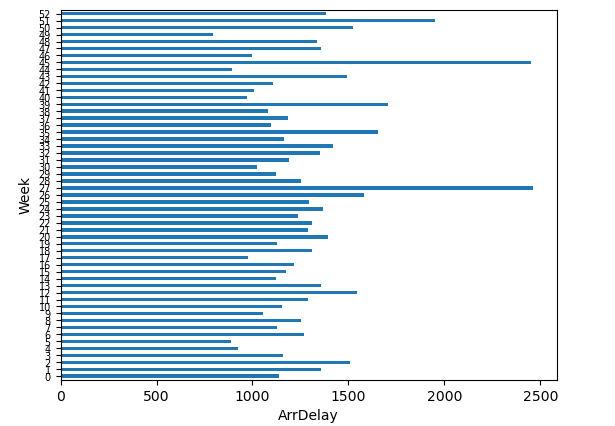
（1）重要数据结果分析

以下列表展示2008年中每一周中的最高抵达延误时间，第一列为周数，第二列为一周中的最高抵达延误时间。由于篇幅有限，此处仅展示部分结果（前50行）。



（2）柱状图表示

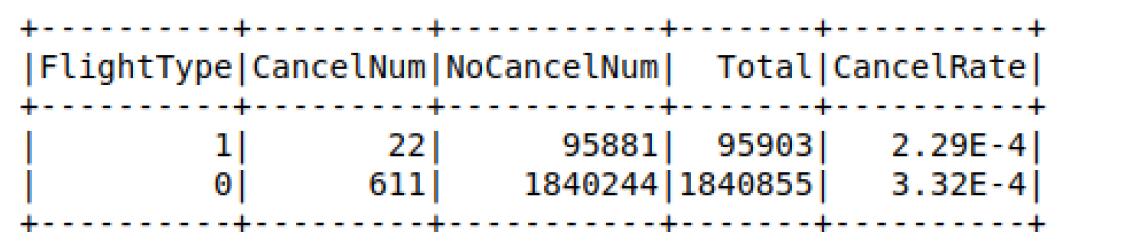
我们将上面的结果转换为可视化表格，调用toPandas()函数将pyspark下的dataframe结构转化为pandas下的dataframe，然后调用plot（kind='barh'）绘制柱状图。柱状图描述2008年每周航班延误峰值，y轴代表的是周数，x轴代表的是每周统计延误时间。柱状图如下图所示：



1. **短途航班和长途航班，哪种航班取消更严重？**

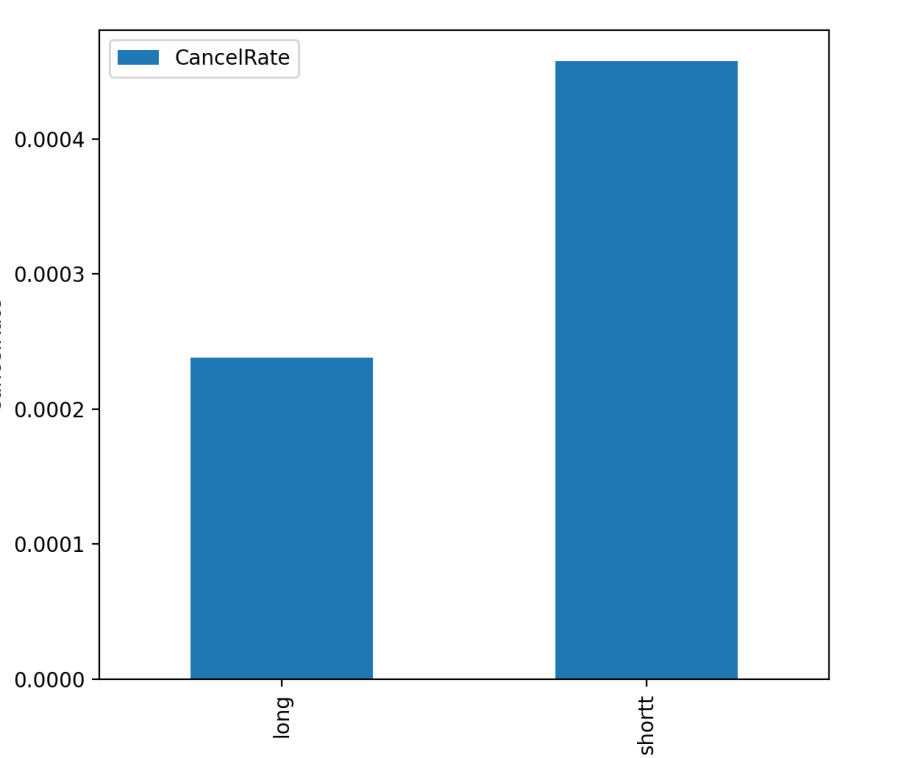
（1）重要数据结果分析

通过运行代码，可以得到以下结果。如图，第一列是航班类型，0为短途航班，1为长途航班，第二列为长途和短途航班分别对应的取消的数目，第三列为长途和短途航班对应没有取消的航班数目，第四列为短途和长途分别的航班总数，第五列为短途和长途航班分别的取消率，根据该结果能够明显看出，短途航班的取消率更高。



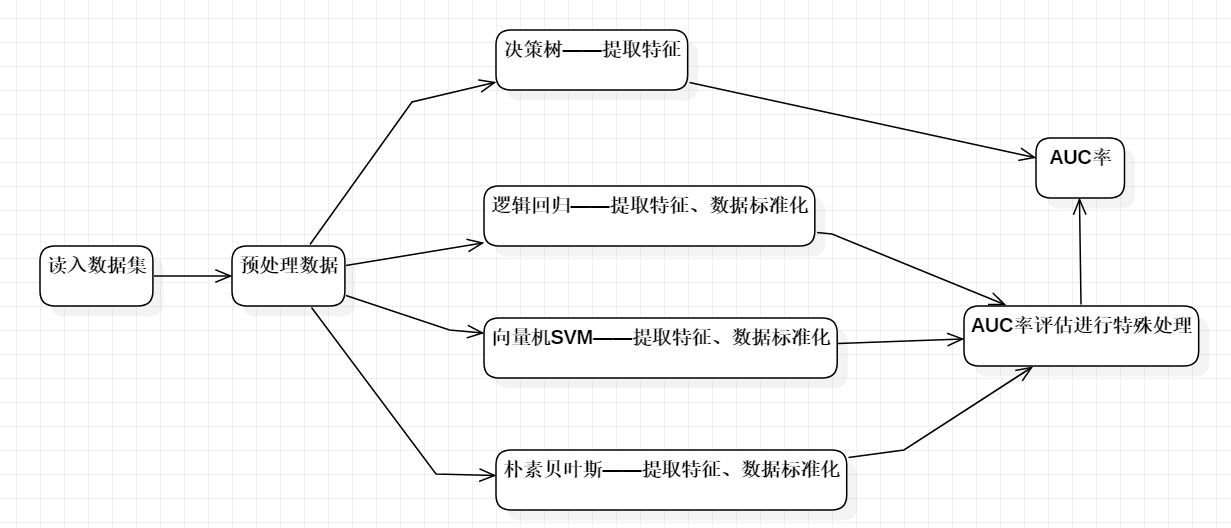
1. 柱状图表示

将结果转换成可视化表格，调用toPandas()函数将pyspark下dataframe结构转换为pandas下的dataframe，然后调用plot(kind=’barh’)绘制柱状图，x轴表示航班类型，y轴表示航班取消率，柱状图如下图所示：

****

1. **建立机器学习算法模型，预测未来航班取消情况。**
2. 核心代码流程分析

代码流程图：



数据预处理代码，提取特征操作，以决策树为例：

import numpy as np

from pyspark.mllib.regression import LabeledPoint

# 数据集路径

global Path

if sc.master[0:5] == "local":

    Path = "file:/hadoop/pythonwork/PythonProject/"

else:

    Path = "hdfs://master:9000/pythonwork/ex-data/"

print("Begin to get data")

# 去除数据集的头，也即数据集中的分类标题，同时以“,”分隔每项数据

rawDataWithHeader = sc.textFile(Path+"ex-data/DelayedFlights.csv")

header = rawDataWithHeader.first()

rawData = rawDataWithHeader.filter(lambda x:x != header)

lines = rawData.map(lambda x:x.split(","))

def extract\_category\_features(field, categoriesMap, num):

    """

    提取字符特征的函数

    :param field: 每项数据

    :param categoriesMap: 字典

    :param num: 索引

    :return:

    """

    categoryIdx = categoriesMap[field[num]]

    categoryFeatures = np.zeros(len(categoriesMap))

    categoryFeatures[categoryIdx] = 1

    return categoryFeatures

def extract\_number\_features(field, num):

    """

    提取数字特征的函数

    :param field: 每项数据

    :param num: 索引

    :return:

    """

    numericalFeatures=[float(field[num])]

    return numericalFeatures

def merge\_features(field):

    """

    将所有特征整合到一起

    :param field: 每项数据

    :return:

    """

    UC\_features = extract\_category\_features(field, UC\_categoriesMap, 9)

    O\_features = extract\_category\_features(field, O\_categoriesMap, 17)

    D\_features = extract\_category\_features(field, D\_categoriesMap, 18)

    Month\_feature = extract\_number\_features(field, 2)

    DayOfMonth\_feature = extract\_number\_features(field, 3)

    DayOfWeek\_feature = extract\_number\_features(field, 4)

    FlightNum\_feature = extract\_number\_features(field, 10)

    Diverted\_feature = extract\_number\_features(field, 24)

    return np.concatenate((UC\_features,O\_features,D\_features, \

                           Month\_feature,DayOfMonth\_feature,DayOfWeek\_feature, \

                           FlightNum\_feature,Diverted\_feature))

# 9-UniqueCarrier 字典

UC\_categoriesMap = lines.map(lambda fields:fields[9]).distinct().zipWithIndex().collectAsMap()

# 17-Origin 字典

O\_categoriesMap = lines.map(lambda fields:fields[17]).distinct().zipWithIndex().collectAsMap()

# 18-Dest 字典

D\_categoriesMap = lines.map(lambda fields:fields[18]).distinct().zipWithIndex().collectAsMap()

def extract\_label(field):

    """

    提取标签 22-cancelled

    :param field:

    :return:

    """

    label = field[22]

    return float(label)

# 创建 LabeledPoint 数据

labelpointRDD = lines.map(lambda r:

                          LabeledPoint(extract\_label(r), merge\_features(r)))

# 以随机方式将数据以 8、1、1 分为 3 个部分

(trainData, validationData, testData) = labelpointRDD.randomSplit([8, 1, 1])

print("将数据分 trainData:" + str(trainData.count()) +

      "   validationData:" + str(validationData.count()) +

      "   testData:" + str(testData.count()))

数据预处理代码，提取特征加标准化操作，以逻辑回归为例：

import sys

from time import time

import pandas as pd

import matplotlib.pyplot as plt

from pyspark import SparkConf, SparkContext

from pyspark.mllib.classification import LogisticRegressionWithSGD

from pyspark.mllib.regression import LabeledPoint

import numpy as np

from pyspark.mllib.evaluation import BinaryClassificationMetrics

from pyspark.mllib.feature import StandardScaler

def extract\_category\_features(field, categoriesMap, num):

    """

    提取字符特征的函数

    :param field: 每项数据

    :param categoriesMap: 字典

    :param num: 索引

    :return:

    """

    categoryIdx = categoriesMap[field[num]]

    categoryFeatures = np.zeros(len(categoriesMap))

    categoryFeatures[categoryIdx] = 1

    return categoryFeatures

def extract\_number\_features(field, num):

    """

    提取数字特征的函数

    :param field: 每项数据

    :param num: 索引

    :return:

    """

    numericalFeatures=[float(field[num])]

    return numericalFeatures

def merge\_features(field):

    """

    将所有特征整合到一起

    :param field:

    :return:

    """

    UC\_features = extract\_category\_features(field, UC\_categoriesMap, 9)

    O\_features = extract\_category\_features(field, O\_categoriesMap, 17)

    D\_features = extract\_category\_features(field, D\_categoriesMap, 18)

    Month\_feature = extract\_number\_features(field, 2)

    DayOfMonth\_feature = extract\_number\_features(field, 3)

    DayOfWeek\_feature = extract\_number\_features(field, 4)

    FlightNum\_feature = extract\_number\_features(field, 10)

    Diverted\_feature = extract\_number\_features(field, 24)

    return np.concatenate((UC\_features,O\_features,D\_features,

                           Month\_feature,DayOfMonth\_feature,DayOfWeek\_feature,

                           FlightNum\_feature,Diverted\_feature))

def extract\_label(field):

    """

    提取标签-22-cancelled

    :param field:

    :return:

    """

    label = field[22]

    return float(label)

# 数据集路径

global Path

if sc.master[0:5] == "local":

    Path = "file:/hadoop/pythonwork/PythonProject/"

else:

    Path = "hdfs://master:9000/pythonwork/ex-data/"

# 1.导入并转换数据

rawDataWithHeader = sc.textFile(Path+"ex-data/DelayedFlights.csv")

header = rawDataWithHeader.first()

rawData = rawDataWithHeader.filter(lambda x:x != header)

lines = rawData.map(lambda x:x.split(","))

# 2.建立训练评估所需数据 RDD[LabeledPoint]

# 9-UniqueCarrier 字典

UC\_categoriesMap = lines.map(lambda fields:fields[9]) \

    .distinct().zipWithIndex().collectAsMap()

# 17-Origin 字典

O\_categoriesMap = lines.map(lambda fields:fields[17]) \

    .distinct().zipWithIndex().collectAsMap()

# 18-Dest 字典

D\_categoriesMap = lines.map(lambda fields:fields[18]) \

    .distinct().zipWithIndex().collectAsMap()

labelRDD = lines.map(lambda r: extract\_label(r))

featureRDD = lines.map(lambda r: merge\_features(r))

# 这里的 withMean 参数为 True，在朴素贝叶斯的模型立这个参数应当设置为 False

stdScaler = StandardScaler(withMean=True, withStd=True).fit(featureRDD)

ScalerFeatureRDD = stdScaler.transform(featureRDD)

labelpoint = labelRDD.zip(ScalerFeatureRDD)

labelpointRDD = labelpoint.map(lambda r: LabeledPoint(r[0], r[1]))

# 3.以随机方式将数据分为 3 个部分并且返回------------

(trainData, validationData, testData) = labelpointRDD.randomSplit([8, 1, 1])

print("将数据分 trainData:" + str(trainData.count()) +

      "   validationData:" + str(validationData.count()) +

      "   testData:" + str(testData.count()))

模型训练代码，以决策树为例：

from pyspark.mllib.evaluation import BinaryClassificationMetrics

def evaluateModel(model, validationData):

    """

    计算模型的 AUC 率

    :param model:

    :param validationData:

    :return:

    """

    score = model.predict(validationData.map(lambda p:p.features))

    scoreAndLabels = score.zip(validationData.map(lambda p:p.label))

    metrics = BinaryClassificationMetrics(scoreAndLabels)

    AUC = metrics.areaUnderROC

    return(AUC)

def evalAllParameter(trainData, validationData,impurityList, maxDepthList, maxBinsList):

    """

    评估模型的所有参数对AUC率的影响并显示出来

    :param trainData:

    :param validationData:

    :param impurityList:

    :param maxDepthList:

    :param maxBinsList:

    :return: 带有最佳参数的最佳模型

    """

    metrics = [trainEvaluateModel(trainData, validationData,

                                  impurity,maxDepth,  maxBins  )

               for impurity in impurityList

               for maxDepth in maxDepthList

               for  maxBins in maxBinsList ]

    Smetrics = sorted(metrics, key=lambda k: k[0], reverse=True)

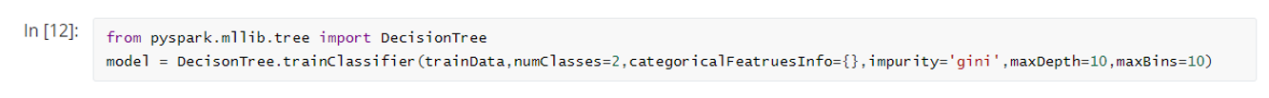
    bestParameter=Smetrics[0]

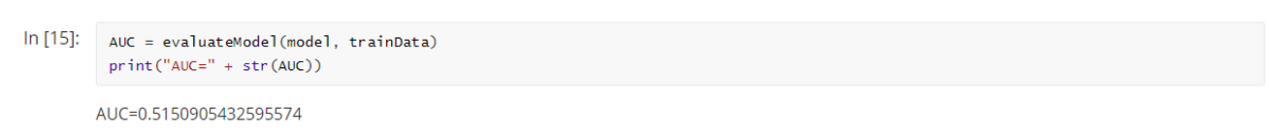
    print("调校后最佳参数：impurity:" + str(bestParameter[2]) + "  ,maxDepth:" + str(bestParameter[3]) +  "  ,maxBins:" + str(bestParameter[4])   + "  ,结果AUC = " + str(bestParameter[0]))

    return bestParameter[5]

1. 重要数据结果分析

决策树模型的训练参数及AUC率：

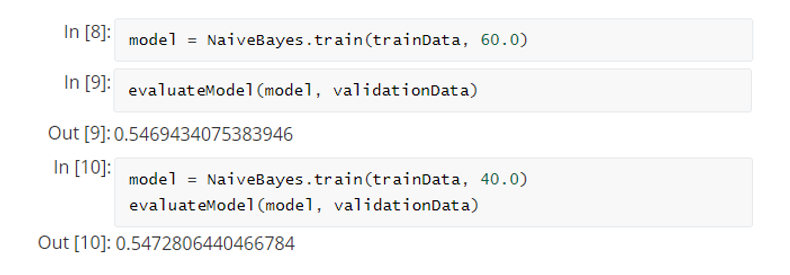




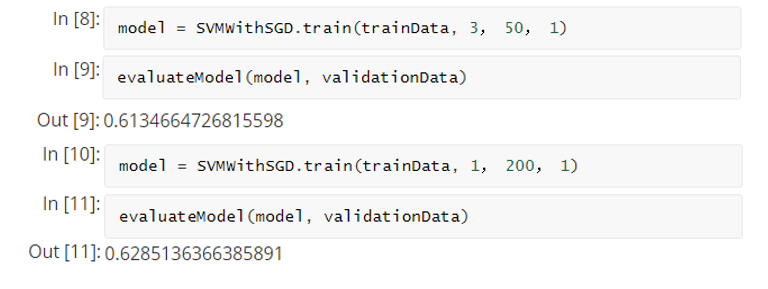
逻辑回归模型的训练参数及AUC率：



朴素贝叶斯模型的训练参数及AUC率：



向量机模型的训练参数及AUC率：



综上所述，当前最佳模型AUC率约为0.66。

# 十一、总结及心得体会

针对本次航空公司延误和取消分析实验，我们首先经过小组会议，共同商议、拆分任务，大家分工合作。其次汇总过程中再进一步细化完善，大家齐心协力克服了在实验中遇到的困难。通过对实验中六个问题的分析探讨到动手操作，最终顺利完成了以上报告，提升了每个人独立思考和交流合作的实践能力。

在完成实验的过程中，我们掌握使用了PySpark来分析数据。基于Python语言进行Spark Application编程，完成数据获取、处理、数据分析及可视化方面常用的数据分析方法与技巧。不仅让我们复习了在理论课堂上学到的知识，同时，纸上得来终觉浅，绝知此事要躬行。我们每个人都在实践中加深了对大数据分析与计算相关内容的理解和新的认识。

在汤老师的理论课堂上我们已经对大数据计算系统的数据模型、处理算法等有了初步认识。但是在李老师的实验课上，我们发现在实际操作过程中，本实验对我们来说并不轻松，具有一定难度。遇到的各种小问题都不可忽视，影响最终结果的成功与否。在大家的互相帮助和交流下，同时我们也运用网络学习了之前不曾遇到的新知识，一步步相互推动，才最终完成实验。

理论与实践的结合，个人思考和团队合作的结合，感谢本次大数据实验，让我们受益匪浅。

# 十二、对本项目过程及方法、手段的改进建议

暂无。