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**

实验教师 **

学生信息：

序号	学号	姓名
1	20210909**	**
2	**	**
3	**	**
4	**	**
5	**	**

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：杨珊　　地点：二教 110

一、 项目名称：航空公司延误和取消分析项目

二、 项目时间：2023.11.22—2023.12.22

三、 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。

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。

对于机器学习模型的选择，可以根据问题的特性和数据集的大小来决定。若选择深度学习方法，LSTM 模型在处理涉及时间序列的数据时通常效果较好。

在建模过程中，需要使用准确率、召回率、ROC 曲线、AUC 等指标来评估

模型性能，以便选择最适合数据集和问题的模型。最终，通过综合分析，可以提取有价值的信息，为航空公司延误和取消提供有效的预测和分析。

四、 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。

2. 任务概述

2.1 目标

本项目需分析航空公司延误和取消情况，以帮助航空公司了解延误和取消的原因，并提供预测模型以预测未来的航班取消情况。并提供数据可视化和统计分析结果，以便决策者和相关人员能够更好地理解和利用数据。项目选取对 DataExpo2009 数据集，通过 Python、PySpark 等工具，进行航空公司延误和取消分析，并在此基础上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。针对数据集的规模和复杂性，考虑使用分布式计算框架进行大数据处理和分析，以提高性能和效率。考虑数据集的增长和未来需求的变化，设计可扩展的架构和算法，以便在需要时能够处理更大规模的数据和更复杂的分析任务。通过数据获取和处理、数据

分析和建模、数据可视化等步骤实现对航空公司延误和取消情况的分析，并提供预测模型以预测未来航班取消情况。

2.2 用户特点

用户特点：范围较广，暂无客观规律性

3. 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。

4. 用例图

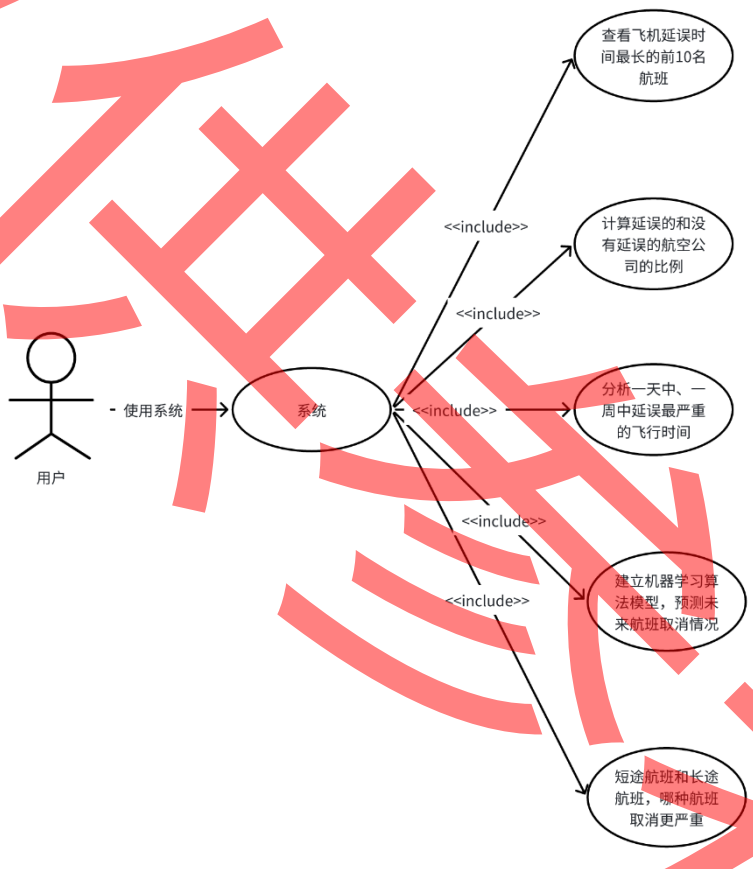


图 1-1 系统用例图

要素	含义与要求
用例名称	查看飞机延误时间最长的前 10 名航班
简要描述	展示飞机延误时间最长的前 10 名航班的相关信息
参与者	用户、系统
前置流程	已经获取并处理了航班数据
基本流程	1. 用户发出查看飞机延误时间最长的前 10 名航班的请求。 2. 系统根据航班延误时间对航班数据进行排序，选取延误时间最长的前 10 名航班。
备选流程	如果航班数据为空或不足 10 条： 系统向用户显示适当的消息，说明无法获取足够的航班数据进行展示。
后置条件	无

系统规约 1-1 查找航班

要素	含义与要求
用例名称	计算延误和没有延误的航空公司的比例
简要描述	计算延误和没有延误的航空公司的比例，并展示结果。
参与者	用户、系统
前置流程	已经获取并处理了航班数据
基本流程	1. 用户发出计算延误和没有延误的航空公司比例的请求。 2. 系统统计延误的航班数量和没有延误的航班数量。 3. 系统计算延误和没有延误的航空公司的比例。
备选流程	如果航班数据为空： 系统向用户显示适当的消息，说明无法计算延误和没有延误的航空公司比例。
后置条件	无

系统规约 1-2 计算延误和没有延误航班

要素	含义与要求
用例名称	分析一天中、一周中延误最严重的飞行时间
简要描述	分析一天中和一周中延误最严重的飞行时间段，并展示结果。
参与者	用户、系统
前置流程	已经获取并处理了航班数据
基本流程	1. 用户发出分析一天中和一周中延误最严重的飞行时间的请求。 2. 系统根据航班数据按照时间维度进行分析，计算一天中和一周中延误最严重的飞行时间段。
备选流程	如果航班数据为空： 系统向用户显示适当的消息，说明无法分析延误最严重的飞行时间。
后置条件	无

系统规约 1-3 分析一天中、一周中延误最严重的飞行时间

要素	含义与要求
用例名称	短途航班和长途航班，哪种航班取消更严重
简要描述	比较短途航班和长途航班的取消情况，确定哪种航班的取消更严重，并展示结果。
参与者	用户、系统
前置流程	已经获取并处理了航班数据
基本流程	1. 用户发出比较短途航班和长途航班取消情况的请求。 2. 系统根据航班数据筛选出短途航班和长途航班，并统计它们的取消数量。 3. 系统比较短途航班和长途航班的取消数量，确定哪种航班的取消更严重。
备选流程	如果航班数据为空： 系统向用户显示适当的消息，说明无法比较短途航班和长途航班的取消情况。
后置条件	无

系统规约 1-4 短途航班和长途航班分析

要素	含义与要求
用例名称	建立机器学习算法模型，预测未来航班取消情况
简要描述	建立机器学习算法模型,使用历史航班数据预测未来航班的取消情况。
参与者	用户、系统
前置流程	已经获取并处理了航班数据
基本流程	1. 用户发出建立机器学习算法模型预测航班取消情况的请求。 2. 系统根据历史航班数据，进行特征选择和模型训练。 3. 系统使用训练好的模型对未来航班数据进行预测，得出航班的取消概率。 4. 系统生成预测结果报告，并展示给用户。
备选流程	如果无法建立机器学习算法模型或模型训练失败：系统向用户显示错误消息,并说明无法预测航班取消情况。
后置条件	无

系统规约 1-5 建立机器学习算法模型

5. 数据流图

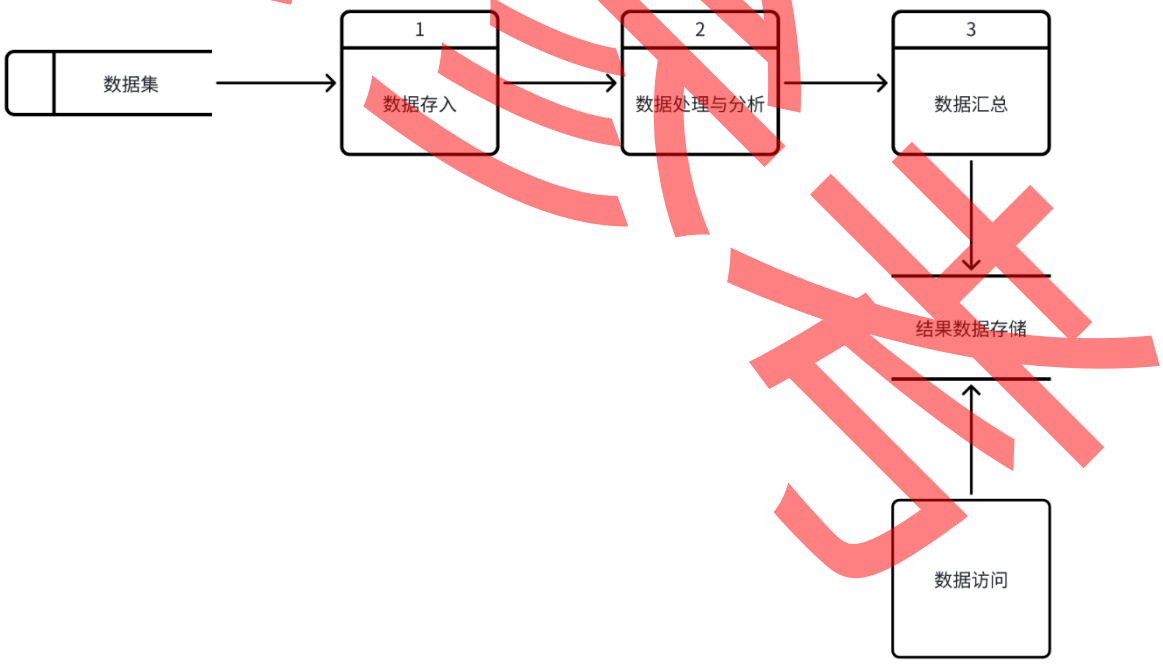


图 1-2 数据流图

6. 顺序图

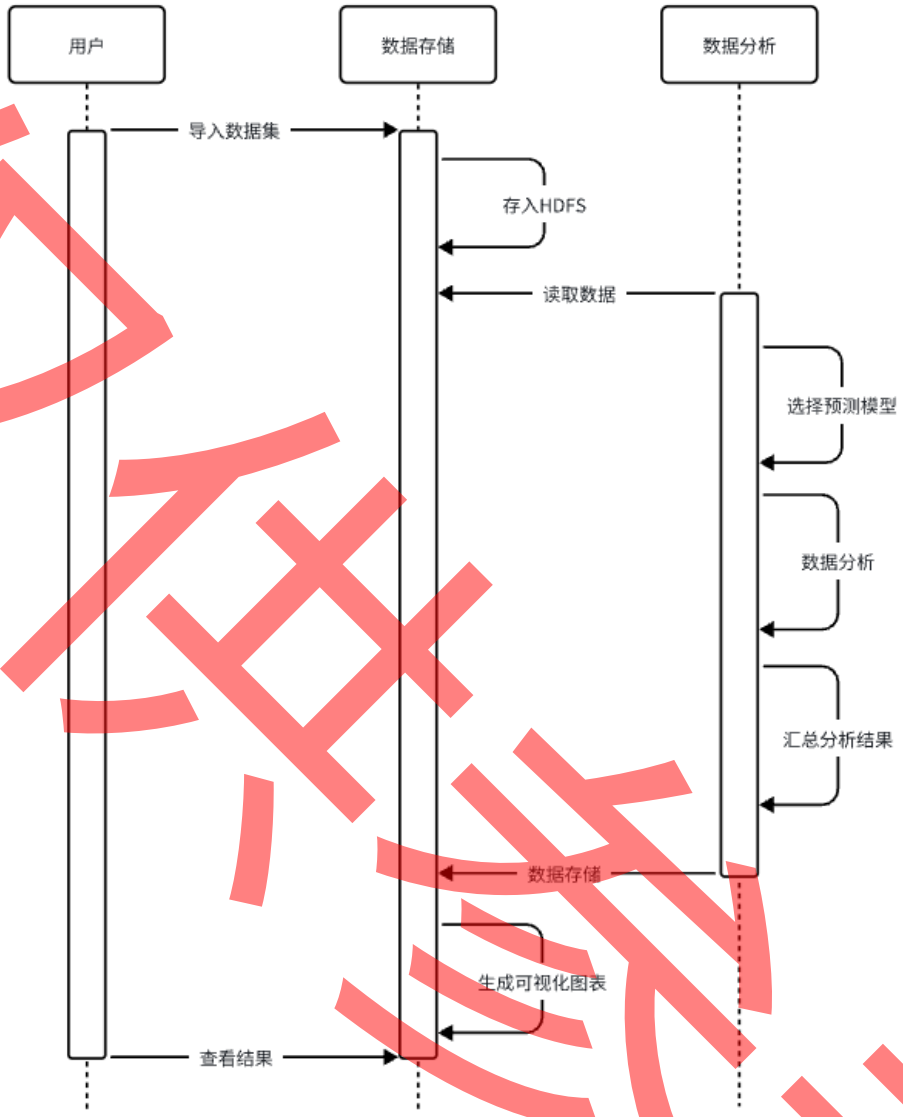


图 1-3 顺序图

7. 流程图

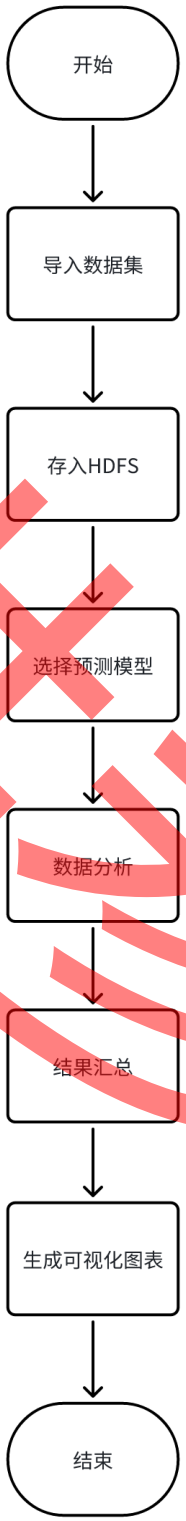


图 1-4 流程图

六、 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

a) 下载 Anaconda 包

```
020.02-Linux-x86_64.sh
--2022-12-07 19:12:34--  https://repo.anaconda.com/archive/Anaconda3-2020.02-Li
nux-x86_64.sh
正在解析主机 repo.anaconda.com (repo.anaconda.com)... 104.16.131.3, 104.16.130.
3, 2606:4700::6810:8303, ...
正在连接 repo.anaconda.com (repo.anaconda.com)|104.16.131.3|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：  546910666 (522M) [application/x-sh]
正在保存至：“Anaconda3-2020.02-Linux-x86_64.sh”

Anaconda3-2020.02-L 100%[==================>] 521.57M  9.96MB/s    用时 59s

2022-12-07 19:13:34 (8.85 MB/s) - 已保存 “Anaconda3-2020.02-Linux-x86_64.sh” [5
46910666/546910666])
```

b) 安装 Anaconda

```
Welcome to Anaconda3 2020.02

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>> ^[^A
```

```
Anaconda3 will now be installed into this location:
/home/hadoop/anaconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/hadoop/anaconda3] >>>
PREFIX=/home/hadoop/anaconda3
Unpacking payload ...
```

c) 验证 python 环境

```
hadoop@hadoop-virtual-machine:/hadoop$ python
Python 3.7.6 (default, Jan  8 2020, 19:59:22)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
Type "help", "copyright", "credits" or "license" for more information.
2022-12-08 00:13:50,061 WARN util.Utils: Your hostname, hadoop-virtual-machine r
esolves to a loopback address: 127.0.1.1; using 192.168.80.129 instead (on inter
face ens33)
2022-12-08 00:13:50,063 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind
to another address
2022-12-08 00:13:50,900 WARN util.NativeCodeLoader: Unable to load native-hadoop
 library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.1.2
      /_/

Using Python version 3.8.5 (default, Jul 28 2020 12:59:40)
Spark context Web UI available at http://192.168.80.129:4040
Spark context available as 'sc' (master = local[*], app id = local-1670429633373
).
SparkSession available as 'spark'.
>>>
```

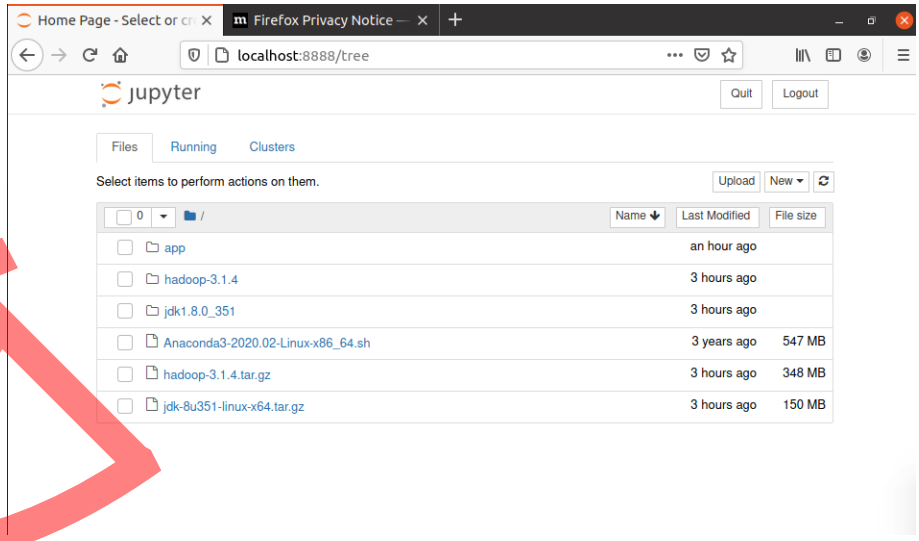
（2） Jupyter 安装配置

a) 配置 PySpark driver

b) 启动 Jupyter

```
op/.local/share/jupyter/runtime/notebook_cookie_secret
[I 00:16:36.274 NotebookApp] JupyterLab extension loaded from /home/hadoop/anaco
nda3/lib/python3.7/site-packages/jupyterlab
[I 00:16:36.274 NotebookApp] JupyterLab application directory is /home/hadoop/an
aconda3/share/jupyter/lab
[I 00:16:36.277 NotebookApp] Serving notebooks from local directory: /hadoop
[I 00:16:36.277 NotebookApp] The Jupyter Notebook is running at:
[I 00:16:36.277 NotebookApp] http://localhost:8888/?token=dcccebe521aa8475bbbb55
67bbb1f094f12de660fefb167c
[I 00:16:36.278 NotebookApp]  or http://127.0.0.1:8888/?token=dcccebe521aa8475bb
bb5567bbb1f094f12de660fefb167c
[I 00:16:36.278 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
[C 00:16:36.446 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/hadoop/.local/share/jupyter/runtime/nbserver-10637-open.htm
l
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=dcccebe521aa8475bbbb5567bbb1f094f12de660fef
b167c
     or http://127.0.0.1:8888/?token=dcccebe521aa8475bbbb5567bbb1f094f12de660fef
b167c
```



八、项目实践过程

1. 准备过程

（1） 启动 Hadoop

```
hadoop@hadoop-virtual-machine:~/hadoop-3.1.4/sbin$ ./start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
```

```
hadoop@hadoop-virtual-machine:~/hadoop-3.1.4/sbin$ jps
2624 ResourceManager
3009 DataNode
2948 NameNode
3046 Jps
```

上图显示启动成功。

（2） 启动 Spark

```
hadoop@hadoop-virtual-machine:~/app/spark/sbin$ ./start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /hadoop/app/spark/log
s/spark-hadoop-org.apache.spark.deploy.master.Master-1-hadoop-virtual-machine.ou
t
```

上图显示启动成功。

（3） 上传文件

将数据集文件导入到 hadoop 用户中：

```
hadoop@hadoop-virtual-machine:~/data$ ls
三国演义.txt        sanguoyanyi.zip  word_zh.txt
DelayedFlights.csv  spark            zhwiki-latest-abstract-zh-cn2.xml
hadoop-data         word.txt
```

将数据集文件上传到 hdfs 系统中：

```
hadoop@hadoop-virtual-machine:~/data$ hadoop fs -mkdir /airport
2023-12-19 11:03:18,323 WARN util.NativeCodeLoader: Unable to load native-hadoop
 library for your platform... using builtin-java classes where applicable
hadoop@hadoop-virtual-machine:~/data$ hadoop fs -put ./DelayedFlights.csv /airpo
rt
2023-12-19 11:03:45,196 WARN util.NativeCodeLoader: Unable to load native-hadoop
 library for your platform... using builtin-java classes where applicable
```

2. 问题一：“查看飞机延误时间最长的前 10 名航班”

（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

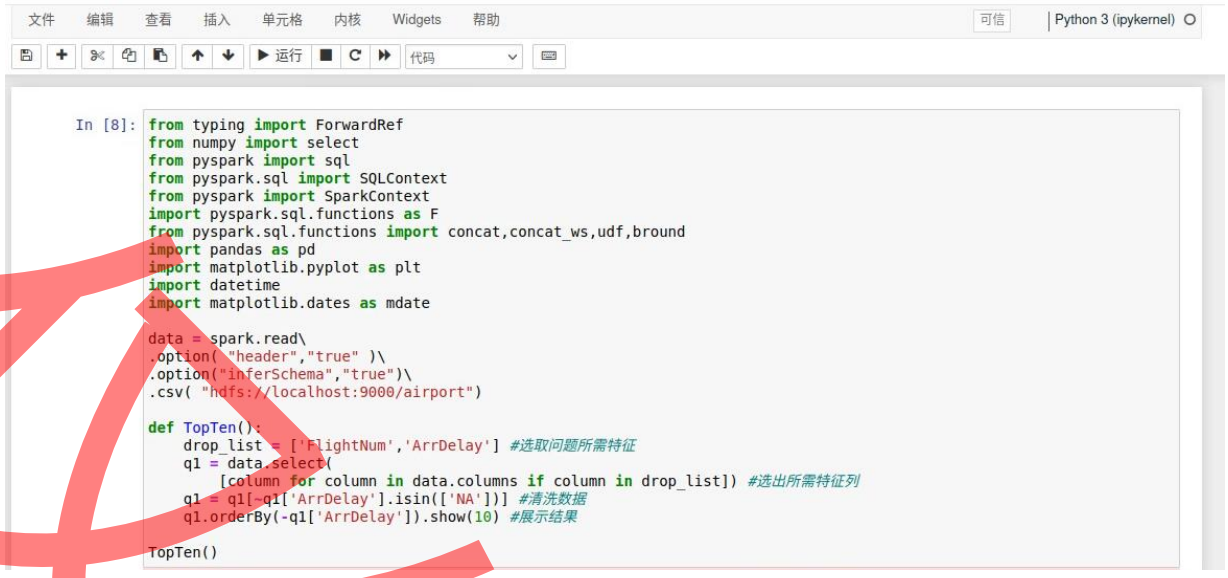
代码 1

```
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：



3. 问题二：“计算延误的和没有延误的航空公司的比例”

（1） 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。

（2） 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('UniqueCarr
ier').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('UniqueC
arrier').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', 'NoDelay
Times') #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', ho
w='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['NoDelayT
imes']).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()
```

4. 问题三：分析一天中、一周中延误最严重的飞行时间”

（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：

```
In [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
    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()
```

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：

In [7]:
```
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
    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
    q3_week.orderBy('weekNum').show(50) #显示五十行结果

    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()
```

5. 问题四：“短途航班和长途航班，哪种航班取消更严重”

（1） 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**。

（2） 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()#
```

（3） 运行界面

6. 问题五：建立机器学习算法模型，预测未来航班取消情况

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：

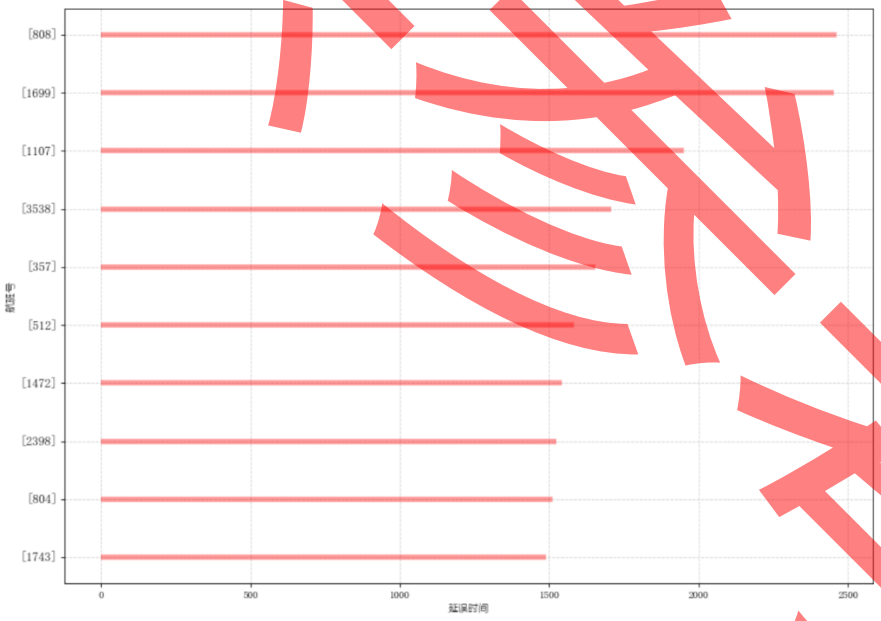
(1)重要数据结果分析

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。

```
+---------+--------+
|FlightNum|ArrDelay|
+---------+--------+
|      808|  2461.0|
|     1699|  2453.0|
|     1107|  1951.0|
|     3538|  1707.0|
|      357|  1655.0|
|      512|  1583.0|
|     1472|  1542.0|
|     2398|  1525.0|
|      804|  1510.0|
|     1743|  1490.0|
+---------+--------+
only showing top 10 rows
```

（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：



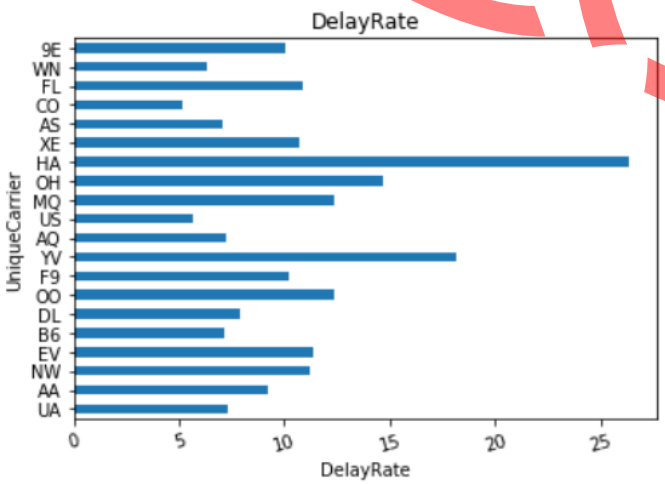
2. 计算延误的和没有延误的航空公司的比例：

(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。

UniqueCarrier	DelayTimes	NoDelayTimes	DelayRate
UA	123989	16915	7.330121194206326
AA	172197	18713	9.201998610591568
NW	72395	6448	11.227512406947891
EV	75170	6592	11.403216019417476
B6	48177	6748	7.139448725548311
DL	100923	12805	7.881530652089028
OO	121942	9838	12.394998983533238
F9	25708	2516	10.217806041335454
YV	63289	3480	18.186494252873562
AQ	654	90	7.266666666666667
US	83262	14745	5.646795523906409
MQ	130647	10576	12.353158093797276
OH	49104	3349	14.66228724992535
HA	7199	273	26.36996336996337
XE	94313	8834	10.67613764998868
AS	34179	4831	7.074932726143656
CO	83646	16085	5.200248678893379
FL	65008	5961	10.905552759604094
WN	324717	51484	6.307143967057726
9E	46896	4673	10.035523218489192

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。

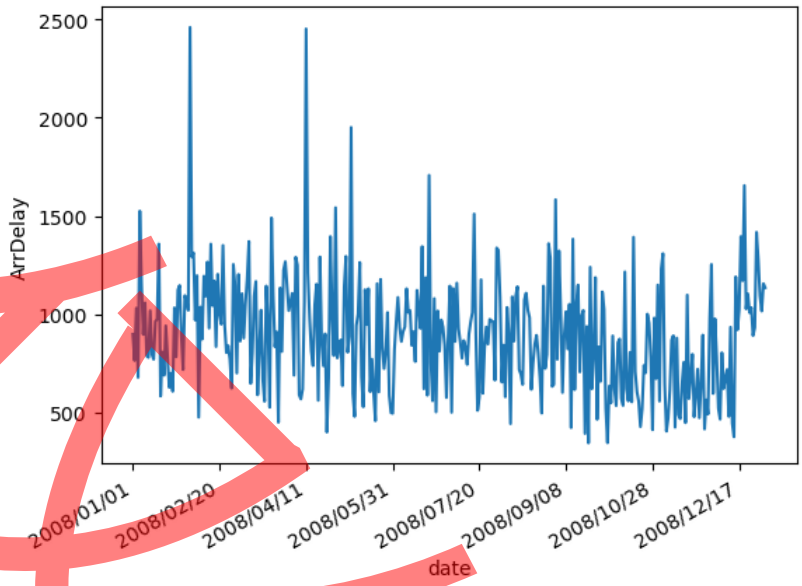
（1）重要数据结果分析

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）。

```
+----+-----+----------+---------+---------------+
|Year|Month|DayofMonth|DayOfWeek|maxDelayInOneDay|
+----+-----+----------+---------+---------------+
|2008|    1|         1|        2|          897.0|
|2008|    1|         2|        3|          763.0|
|2008|    1|         3|        4|         1030.0|
|2008|    1|         4|        5|          677.0|
|2008|    1|         5|        6|         1525.0|
|2008|    1|         6|        7|         1147.0|
|2008|    1|         7|        1|          896.0|
|2008|    1|         8|        2|         1057.0|
|2008|    1|         9|        3|          786.0|
|2008|    1|        10|        4|          779.0|
|2008|    1|        11|        5|         1017.0|
|2008|    1|        12|        6|          794.0|
|2008|    1|        13|        7|          769.0|
|2008|    1|        14|        1|          963.0|
|2008|    1|        15|        2|          973.0|
|2008|    1|        16|        3|         1357.0|
|2008|    1|        17|        4|          582.0|
|2008|    1|        18|        5|          807.0|
|2008|    1|        19|        6|          688.0|
|2008|    1|        20|        7|          940.0|
|2008|    1|        21|        1|          785.0|
|2008|    1|        22|        2|          626.0|
|2008|    1|        23|        3|          699.0|
|2008|    1|        24|        4|          606.0|
|2008|    1|        25|        5|         1032.0|
|2008|    1|        26|        6|          782.0|
|2008|    1|        27|        7|         1123.0|
|2008|    1|        28|        1|         1146.0|
|2008|    1|        29|        2|          888.0|
|2008|    1|        30|        3|          717.0|
|2008|    1|        31|        4|         1094.0|
|2008|    2|         1|        5|         1064.0|
|2008|    2|         2|        6|         1019.0|
|2008|    2|         3|        7|         2461.0|
|2008|    2|         4|        1|         1292.0|
|2008|    2|         5|        2|         1312.0|
|2008|    2|         6|        3|          969.0|
|2008|    2|         7|        4|         1196.0|
|2008|    2|         8|        5|          474.0|
|2008|    2|         9|        6|         1036.0|
|2008|    2|        10|        7|          872.0|
|2008|    2|        11|        1|         1193.0|
|2008|    2|        12|        2|         1091.0|
|2008|    2|        13|        3|         1265.0|
|2008|    2|        14|        4|          928.0|
|2008|    2|        15|        5|         1357.0|
|2008|    2|        16|        6|         1043.0|
|2008|    2|        17|        7|         1169.0|
|2008|    2|        18|        1|          834.0|
|2008|    2|        19|        2|         1203.0|
+----+-----+----------+---------+---------------+
only showing top 50 rows
```

（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。

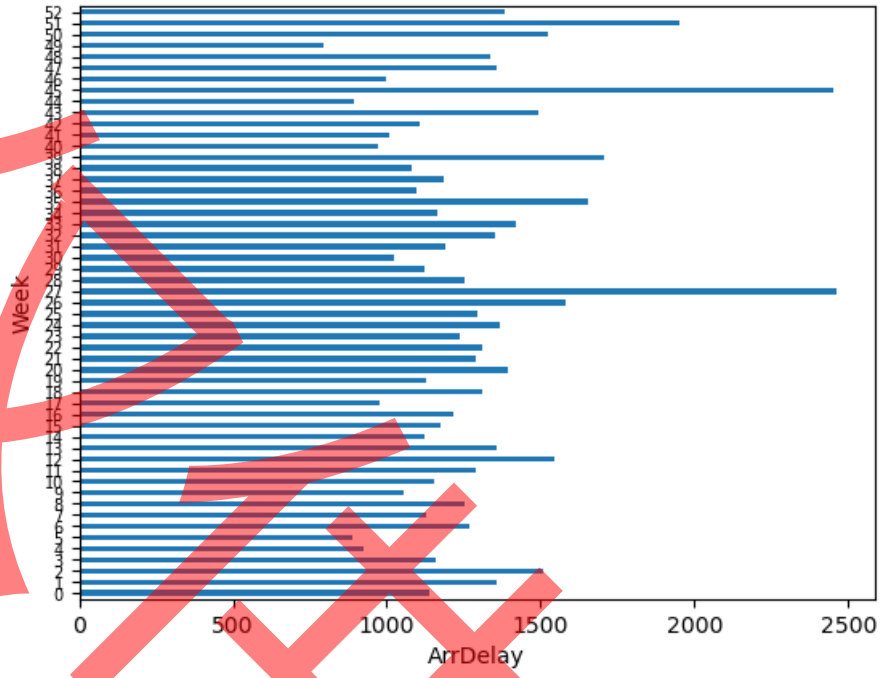
weekNum	maxDelayInOneWeek
0	1525.0
1	1057.0
2	1357.0
3	1123.0
4	2461.0
5	1312.0
6	1357.0
7	1350.0
8	1253.0
9	1370.0
10	1166.0
11	1490.0
12	1267.0
13	1290.0
14	2453.0
15	1291.0
16	1542.0
17	1294.0
18	1951.0
19	1129.0
20	1177.0
21	1008.0
22	1129.0
23	1121.0
24	1707.0
25	1079.0
26	1158.0
27	926.0
28	1510.0
29	973.0
30	1337.0
31	1140.0
32	1107.0
33	892.0
34	1359.0
35	1583.0
36	1382.0
37	1239.0
38	1186.0
39	1025.0
40	1215.0
41	1392.0
42	1000.0
43	1308.0
44	890.0
45	1097.0
46	795.0
47	1253.0
48	976.0
49	1190.0

only showing top 50 rows

（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：



5. 短途航班和长途航班，哪种航班取消更严重？

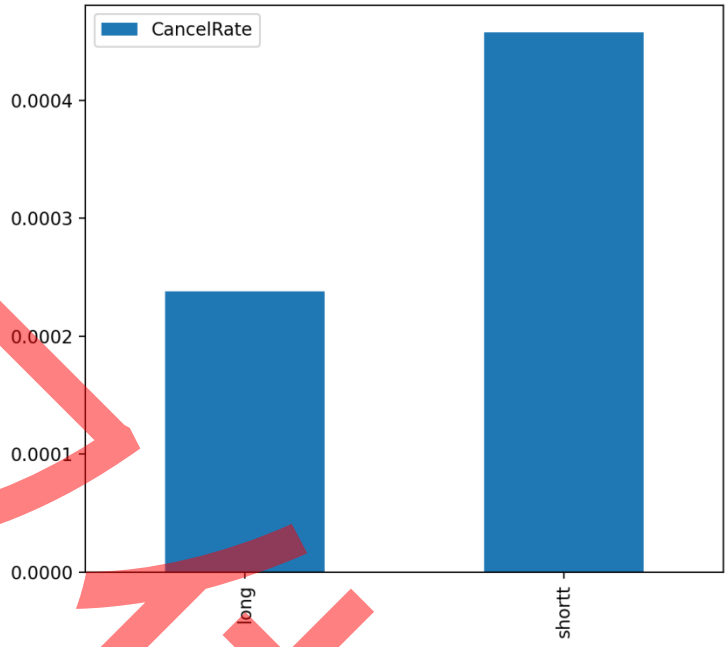
（1）重要数据结果分析

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。

FlightType	CancelNum	NoCancelNum	Total	CancelRate
1	22	95881	95903	2.29E-4
0	611	1840244	1840855	3.32E-4

（2） 柱状图表示

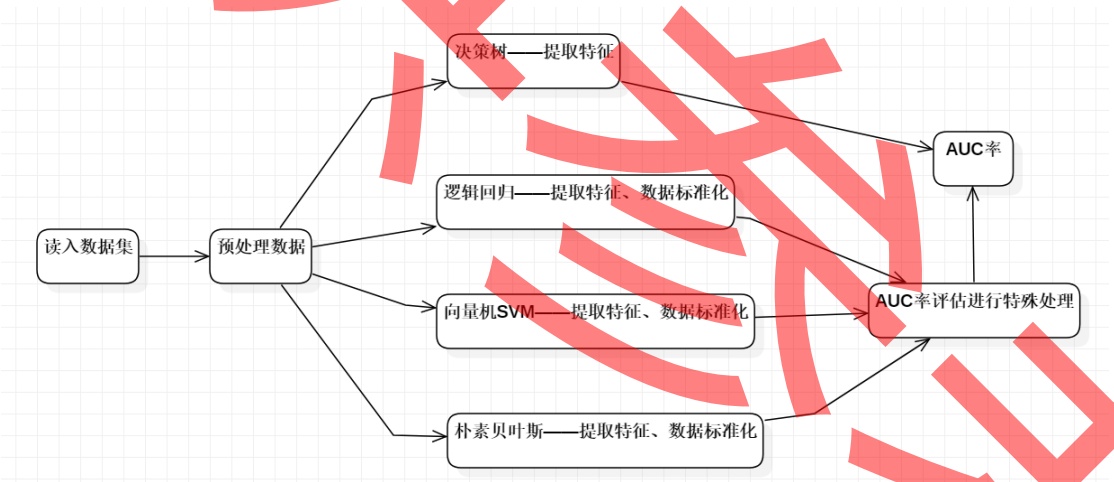
将结果转换成可视化表格，调用 toPandas()函数将 pyspark 下 dataframe 结构转换为 pandas 下的 dataframe，然后调用 plot(kind=’barh’)绘制柱状图，x 轴表示航班类型，y 轴表示航班取消率，柱状图如下图所示：



6. 建立机器学习算法模型，预测未来航班取消情况。

（1） 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,DayOfWe
ek_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):
        categoryIdx = categoriesMap[field[num]]
        categoryFeatures = np.zeros(len(categoriesMap))
        categoryFeatures[categoryIdx] = 1
        return categoryFeatures

    def extract_number_features(field, num):
        numericalFeatures=[float(field[num])]
        return numericalFeatures

    def merge_features(field):
        """
        将所有特征整合到一起
        """
        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,DayOfWe
ek_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]
```

（2） 重要数据结果分析

决策树模型的训练参数及 AUC 率：

In [12]:
```
from pyspark.mllib.tree import DecisionTree
model = DecisonTree.trainClassifier(trainData,numClasses=2,categoricalFeatruesInfo={},impurity='gini',maxDepth=10,maxBins=10)
```

In [15]:
```
AUC = evaluateModel(model, trainData)
print("AUC=" + str(AUC))
```

```
AUC=0.5150905432595574
```

逻辑回归模型的训练参数及 AUC 率：

In [ ]:
```
model = evalAllParameter(trainData,validationData,[3,5,10,15],[10,50,100],[0.5,0.8,1])
```

```
训练评估：使用参数 numIteration=3 stepSize=10 miniBatchFraction=0.5 所需时间=400.16985711472576 结果AUC = 0.6187587227
训练评估：使用参数 numIteration=3 stepSize=10 miniBatchFraction=0.8 所需时间=387.84599774412555 结果AUC = 0.6317252728
训练评估：使用参数 numIteration=3 stepSize=10 miniBatchFraction=1 所需时间=396.27527171717175 结果AUC = 0.5978125887
训练评估：使用参数 numIteration=3 stepSize=50 miniBatchFraction=0.8 所需时间=384.75728435872727 结果AUC = 0.5985272177
训练评估：使用参数 numIteration=3 stepSize=50 miniBatchFraction=1 所需时间=395.12278575395174 结果AUC = 0.64460278215
训练评估：使用参数 numIteration=3 stepSize=50 miniBatchFraction=0.5 所需时间=422.4693845685230 结果AUC = 0.574989657780
训练评估：使用参数 numIteration=3 stepSize=100 miniBatchFraction=0.8 所需时间=392.65232686113840 结果AUC = 0.589657780498
训练评估：使用参数 numIteration=3 stepSize=100 miniBatchFraction=1 所需时间=413.64520169385686 结果AUC = 0.657544989765
训练评估：使用参数 numIteration=3 stepSize=100 miniBatchFraction=0.5 所需时间=392.93042664862456 结果AUC = 0.601637980494
训练评估：使用参数 numIteration=5 stepSize=10 miniBatchFraction=0.5 所需时间=420.28653426406196 结果AUC = 0.645255778478
训练评估：使用参数 numIteration=5 stepSize=10 miniBatchFraction=0.8 所需时间=438.14719630686264 结果AUC = 0.635989765785
训练评估：使用参数 numIteration=5 stepSize=10 miniBatchFraction=1 所需时间=430.56934686640169 结果AUC = 0.630358976571
训练评估：使用参数 numIteration=5 stepSize=50 miniBatchFraction=0.8 所需时间=439.40616368645696 结果AUC = 0.659897656804
训练评估：使用参数 numIteration=5 stepSize=50 miniBatchFraction=1 所需时间=391.64066723865219 结果AUC = 0.608989765436
训练评估：使用参数 numIteration=5 stepSize=50 miniBatchFraction=0.5 所需时间=382.04456123616938 结果AUC = 0.614485989765
训练评估：使用参数 numIteration=5 stepSize=100 miniBatchFraction=0.8 所需时间=377.4868612401665 结果AUC = 0.602495989765
训练评估：使用参数 numIteration=5 stepSize=100 miniBatchFraction=1 所需时间=378.86696321640658 结果AUC = 0.593498976567
训练评估：使用参数 numIteration=5 stepSize=100 miniBatchFraction=0.5 所需时间=380.24861690643671 结果AUC = 0.600498976578
训练评估：使用参数 numIteration=10 stepSize=10 miniBatchFraction=0.5 所需时间=449.66938015239686 结果AUC = 0.370129789869
训练评估：使用参数 numIteration=10 stepSize=10 miniBatchFraction=0.8 所需时间=464.71456258664068 结果AUC = 0.373426589775
训练评估：使用参数 numIteration=10 stepSize=10 miniBatchFraction=1 所需时间=470.84586401693669 结果AUC = 0.37514598976
训练评估：使用参数 numIteration=10 stepSize=50 miniBatchFraction=0.8 所需时间=453.14589686634067 结果AUC = 0.342100498974
训练评估：使用参数 numIteration=10 stepSize=50 miniBatchFraction=1 所需时间=461.16523806606893 结果AUC = 0.387560989765
训练评估：使用参数 numIteration=10 stepSize=50 miniBatchFraction=0.5 所需时间=456.3845612369669 结果AUC = 0.360798976543
训练评估：使用参数 numIteration=10 stepSize=100 miniBatchFraction=0.8 所需时间=444.4252866864046 结果AUC = 0.374984298659
训练评估：使用参数 numIteration=10 stepSize=100 miniBatchFraction=1 所需时间=459.13624068646915 结果AUC = 0.384219897753
训练评估：使用参数 numIteration=10 stepSize=100 miniBatchFraction=0.5 所需时间=449.56316922816947 结果AUC = 0.375468676589
训练评估：使用参数 numIteration=15 stepSize=10 miniBatchFraction=0.5 所需时间=521.82560686937164 结果AUC = 0.62039897641
训练评估：使用参数 numIteration=15 stepSize=10 miniBatchFraction=0.8 所需时间=535.95426861643086 结果AUC = 0.649698976578
训练评估：使用参数 numIteration=15 stepSize=10 miniBatchFraction=1 所需时间=534.94126861605806 结果AUC = 0.645114897657
训练评估：使用参数 numIteration=15 stepSize=50 miniBatchFraction=0.8 所需时间=519.456638328886169 结果AUC = 0.695329897647
训练评估：使用参数 numIteration=15 stepSize=50 miniBatchFraction=1 所需时间=530.80646456632186 结果AUC = 0.595862876579
```

朴素贝叶斯模型的训练参数及 AUC 率：

In [8]:
```
model = NaiveBayes.train(trainData, 60.0)
```

In [9]:
```
evaluateModel(model, validationData)
```

Out [9]: 0.5469434075383946

In [10]:
```
model = NaiveBayes.train(trainData, 40.0)
evaluateModel(model, validationData)
```

Out [10]: 0.5472806440466784

向量机模型的训练参数及 AUC 率：

```
In [8]: model = SVMWithSGD.train(trainData, 3, 50, 1)
In [9]: evaluateModel(model, validationData)
Out [9]: 0.6134664726815598
In [10]: model = SVMWithSGD.train(trainData, 1, 200, 1)
In [11]: evaluateModel(model, validationData)
Out [11]: 0.6285136366385891
```

综上所述，当前最佳模型 AUC 率约为 0.66。

十、总结及心得体会

在本次航空公司延误和取消分析实验中，我们小组展开了积极而有序的合作。通过小组会议，我们充分商议并分工合作，拆分任务，为实验的顺利进行奠定了基础。在合作过程中，我们共同克服了遇到的各种困难，通过细化和完善的汇总过程，进一步提高了报告的质量。

实验的关键在于对大数据分析工具的灵活运用，我们选择了使用 PySpark 进行数据分析。通过使用 Python 进行 Spark Application 编程，我们成功地完成了数据获取、处理、分析和可视化等步骤。这使得我们不仅复习了在理论课上学到的知识，而且在实践中加深了对大数据分析与计算相关内容的理解。

在实验的过程中，我们克服了各种小问题，这些问题的解决对最终结果的成功至关重要。我们在相互帮助和交流中，通过网络学习了一些新知识，逐步提高了团队的整体水平。

理论与实践的结合是本次实验的一大特点。在理论课堂上，我们学到了大数据计算系统的数据模型和处理算法等知识。然而，实际操作中，我们发现实验并不轻松，具有一定难度。通过个人思考和团队合作，我们成功地克服了这些难题，最终完成了实验。

在这个过程中，我们不仅提升了个人独立思考和交流合作的实践能力，还收获了对大数据分析与计算领域的深刻认识。感谢本次实验，让我们在理论与实践的交融中受益匪浅。

十一、对本项目过程及方法、手段的改进建议

可以进一步尝试使用深度学习模型，如 Transformer 等大参数量模型，以提高对未来航班取消情况的准确预测能力