

电子科技大学信息与软件工程学院

项 目 报 告

课程名称 大数据分析智能计算

理论教师 罗瑜

实验教师 罗瑜

学生信息：

序号	学号	姓名
1	*****	*****
2	*****	*****
3	*****	*****
4	*****	*****
5	*****	*****
6	*****	*****

本文档为 PPT 六个任务地全部整体实现

八、项目实践过程

本部分仅展示相关代码及运行过程截图。针对每个问题的可视化输出与数据分析、解释、说明将在第九部分阐述。

8.1 读取数据

首先将数据集拷贝至工作目录中

```
$ cp DelayedFlights.csv /hadoop/anaconda3/pythonwork/ipynotebook/
```

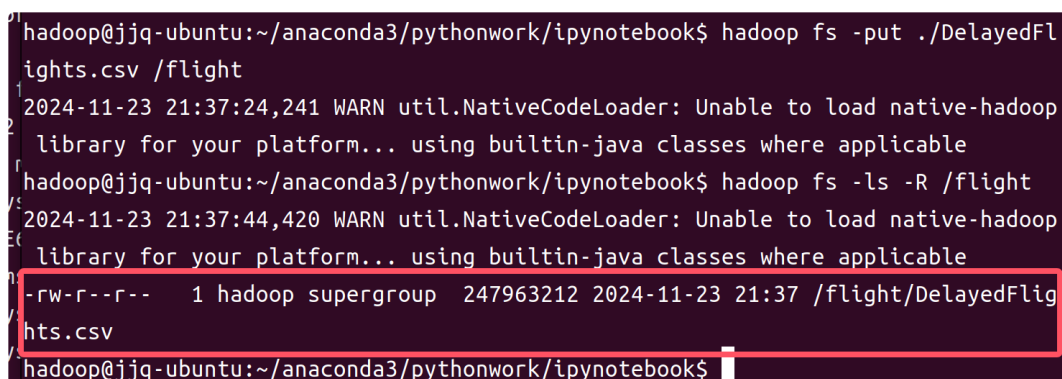
然后将数据拷贝到 HDBS/flight 文件夹下

相关命令为——

```
hadoop fs -mkdir /flight
```

```
hadoop fs -put ./DelayedFlights.csv /flight
```

```
hadoop fs -ls -R /flight
```



```
hadoop@jjq-ubuntu:~/anaconda3/pythonwork/ipynotebook$ hadoop fs -put ./DelayedFlights.csv /flight
2024-11-23 21:37:24,241 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hadoop@jjq-ubuntu:~/anaconda3/pythonwork/ipynotebook$ hadoop fs -ls -R /flight
2024-11-23 21:37:44,420 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
-rw-r--r--  1 hadoop supergroup  247963212 2024-11-23 21:37 /flight/DelayedFlights.csv
hadoop@jjq-ubuntu:~/anaconda3/pythonwork/ipynotebook$
```

从运行截图可知，数据集已经正确上传至 HDBS 中。

从数据集中读取数据。代码如下所示。

代码 1：读取数据集代码

```
from numpy import select
from pyspark import sql
from pyspark.sql import SQLContext
from pyspark import SparkContext
import pyspark.sql.functions as F
from typing import ForwardRef
from pyspark.sql.functions import concat,concat_ws,bround
import pandas as pd
import matplotlib.pyplot as plt
import datetime
```

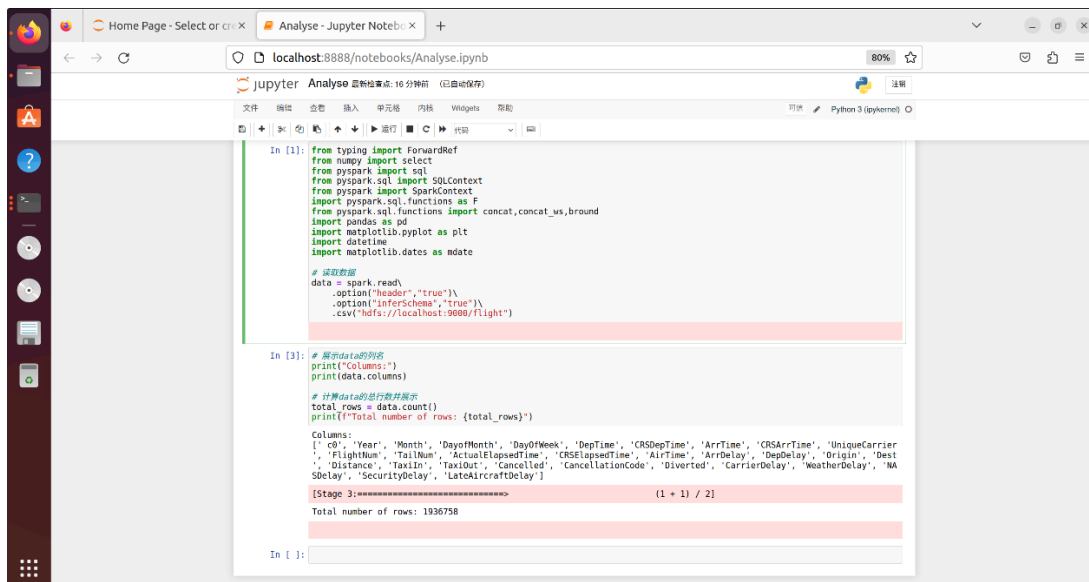
```
import matplotlib.dates as mdate

# 读取数据
data = spark.read\
    .option("header","true")\
    .option("inferSchema","true")\
    .csv("hdfs://localhost:9000/flight")

# 展示data的列名
print("Columns:")
print(data.columns)

# 计算data的总行数并展示
total_rows = data.count()
print(f"Total number of rows: {total_rows}")
```

结果如图：



从截图来看，我们成功读取到了存储在 HDFS 的数据共计 1936758 条。

8.2 查看飞机延误时间最长的前 10 名航班

为了对航班延误情况进行分析，我们依据“抵达延误时间”（即'ArrDelay'字段）对航班数据进行排序。在此过程中，我们将提取两个关键特征：航班号（FlightNum）和抵达延误时间（ArrDelay）。随后，我们将这些数据按照抵达延误时间从大到小的顺序进行排列，并展示排序结果中的前 10 条记录。然后，通过绘图工具 matplotlib 绘制了一个柱状图以可视化呈现。代码如下所示：

代码 2：查看飞机延误时间最长的前 10 名航班

```
import numpy as np
```

```

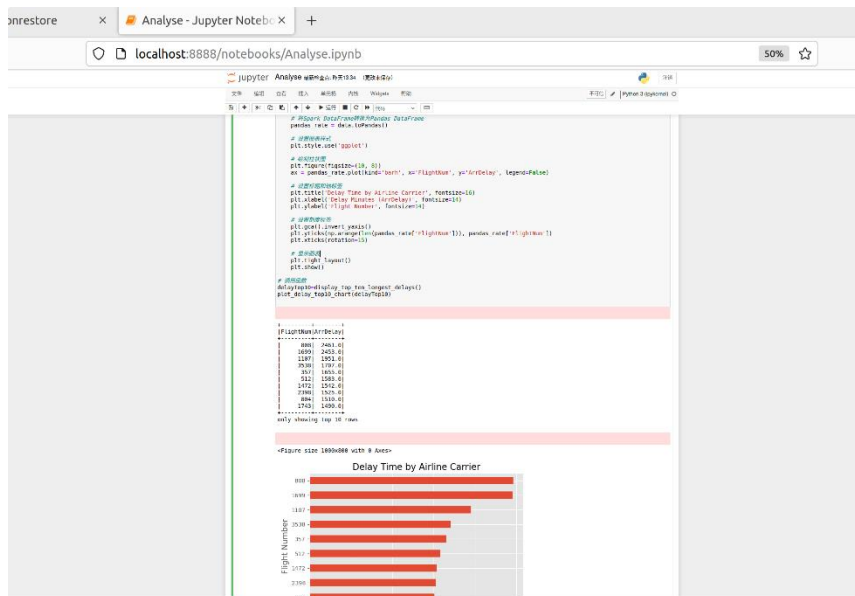
def display_top_ten_longest_delays():
    """
    Task 1:显示延误时间最长的前 10 个航班的函数。
    """
    # 定义需要展示的列名列表
    required_columns = ['FlightNum', 'ArrDelay']
    # 选择所需的列
    selected_data = data.select([column for column in data.columns if column in
required_columns])
    # 按到达延误时间降序排列并显示前 10 条记录
    selected_data.orderBy(selected_data['ArrDelay'].desc()).show(10)
    return selected_data.orderBy(selected_data['ArrDelay'].desc()).limit(10)

def plot_delay_top10_chart(data):
    """
    展示每个航空公司的延误时间 top10。
    """
    # 将 Spark DataFrame 转换为 Pandas DataFrame
    pandas_rate = data.toPandas()
    # 设置图表样式
    plt.style.use('ggplot')
    # 绘制柱状图
    plt.figure(figsize=(10, 8))
    ax = pandas_rate.plot(kind='barh', x='FlightNum', y='ArrDelay', legend=False)
    # 设置标题和轴标签
    plt.title('Delay Time by Airline Carrier', fontsize=16)
    plt.xlabel('Delay Minutes (ArrDelay)', fontsize=14)
    plt.ylabel('Flight Number', fontsize=14)
    # 设置刻度标签
    plt.gca().invert_yaxis()
    plt.yticks(np.arange(len(pandas_rate['FlightNum'])), pandas_rate['FlightNum'])
    plt.xticks(rotation=15)
    # 显示图表
    plt.tight_layout()
    plt.show()

# 调用函数
delayTop10=display_top_ten_longest_delays()
plot_delay_top10_chart(delayTop10)

```

执行结果如图，红色部分为 spark 的 stage 划分日志信息，可忽略。



8.3 计算延误的和没有延误的航空公司的比例

为了计算延误的和没有延误的航空公司的比例。我们只需要统计有延误的航司（即存在 `ArrDelay>0`）数量，然后用总数减去有延误的航司数量，就可以得到无延误的航司数量，进一步可以得到比例。相关代码如下——

代码 3：计算延误的和没有延误的航空公司的比例

```
from pyspark.sql.functions import col
def calculate_delay_stats():
    """
    Task2: 计算有延误和没有延误的航空公司数量及其比例。
    """
    # 计算有延误的航空公司数量
    delay_airlines_count = data.filter(col("ArrDelay") >
0).select("UniqueCarrier").distinct().count()
    # 计算总航空公司数量
    total_airlines_count = data.select("UniqueCarrier").distinct().count()
    # 计算没有延误的航空公司数量
    no_delay_airlines_count = total_airlines_count - delay_airlines_count
    # 计算比例
    if no_delay_airlines_count > 0:
        ratio = delay_airlines_count / no_delay_airlines_count
    else:
        ratio = float('inf') # 如果没有未延误的航空公司，则比例是无限大
    # 返回结果
    return {
        "delay_airlines_count": delay_airlines_count,
        "no_delay_airlines_count": no_delay_airlines_count,
```

```

        "ratio": ratio
    }
# 假设 data 是已经存在的 DataFrame
# 调用函数并打印结果
stats = calculate_delay_stats()
print(f"Number of airlines with delay: {stats['delay_airlines_count']}")
print(f"Number of airlines without delay: {stats['no_delay_airlines_count']}")
print(f"The proportion of delayed to non-delayed airlines: {stats['ratio']:.2f}")

```

```

In [11]: from pyspark.sql.functions import col
def calculate_delay_stats():
    """
    Task2: 计算有延误和没有延误的航空公司数量及其比例。
    """
    # 计算有延误的航空公司数量
    delay_airlines_count = data.filter(col("ArrDelay") > 0).select("UniqueCarrier").distinct().count()

    # 计算总航空公司数量
    total_airlines_count = data.select("UniqueCarrier").distinct().count()

    # 计算没有延误的航空公司数量
    no_delay_airlines_count = total_airlines_count - delay_airlines_count

    # 计算比例
    if no_delay_airlines_count > 0:
        ratio = delay_airlines_count / no_delay_airlines_count
    else:
        ratio = float('inf') # 如果没有未延误的航空公司，则比例是无限大

    # 返回结果
    return {
        "delay_airlines_count": delay_airlines_count,
        "no_delay_airlines_count": no_delay_airlines_count,
        "ratio": ratio
    }

# 假设 data 是已经存在的 DataFrame
# 调用函数并打印结果
stats = calculate_delay_stats()
print(f"Number of airlines with delay: {stats['delay_airlines_count']}")
print(f"Number of airlines without delay: {stats['no_delay_airlines_count']}")
print(f"The proportion of delayed to non-delayed airlines: {stats['ratio']:.2f}")

[Stage 20:> (0 + 2) / 2]
Number of airlines with delay: 20
Number of airlines without delay: 0
The proportion of delayed to non-delayed airlines: inf

```

但是我们发现，事实上，所有航司或多或少都存在延误的航班，不存在完全没有延误的航司。因此求比例无意义。为了更好探究这个问题，我们进一步探究每个公司延误的和没有延误的航班比例。

为了计算每个公司延误的和没有延误的比例，我们将关注于两个关键特征：航空公司的唯一编码（UniqueCarrier）和航班的抵达延误时间（ArrDelay）。通过判断 ArrDelay 字段值是否大于零来确定航班是否延误。

基于这一标准，我们将对数据进行分类统计，并计算出每个航空公司延误航班与未延误航班的次数，然后将延误次数与没有延误次数相除，即可得到比例。

代码 4：计算每个航司延误的和没有延误的航班比例并绘图

```

import numpy as np
def calculate_delay_rate():
    """
    Task2: 计算每个航空公司的延误的和没有延误的比率。
    """

```

```

# 定义需要的列
required_columns = ['UniqueCarrier', 'ArrDelay']

# 选择需要的列
selected_data = data.select([column for column in data.columns if column in
required_columns])

# 计算每个航空公司的延误航班数
delay_counts = selected_data.filter(selected_data['ArrDelay'] >
0).groupBy('UniqueCarrier').count()
delay_counts = delay_counts.withColumnRenamed('count', 'DelayCounts')

# 计算每个航空公司的没有延误的航班数
not_delay_counts = selected_data.filter(selected_data['ArrDelay'] <=
0).groupBy('UniqueCarrier').count()
not_delay_counts = not_delay_counts.withColumnRenamed('count',
'NotDelayCounts')

# 将延误航班数和没有延误的航班数连接起来
joined_data = delay_counts.join(not_delay_counts, on='UniqueCarrier',
how='left_outer')

# 计算延误率
rate_data = joined_data.withColumn('Rate', (joined_data['DelayCounts'] /
joined_data['NotDelayCounts']))

return rate_data

def plot_delay_rate_chart(rate_data):
    """
    展示每个航空公司的延误的和没有延误的比率的柱状图。
    """
    # 将 Spark DataFrame 转换为 Pandas DataFrame
    pandas_rate = rate_data.toPandas()

    # 设置图表样式
    plt.style.use('ggplot')

    # 绘制柱状图
    plt.figure(figsize=(10, 8))
    ax = pandas_rate.plot(kind='barh', x='UniqueCarrier', y='Rate', legend=False)

    # 设置标题和轴标签
    plt.title('Delay Rate by Airline Carrier', fontsize=16)
    plt.xlabel('Rate', fontsize=14)
    plt.ylabel('Airline Carrier', fontsize=14)

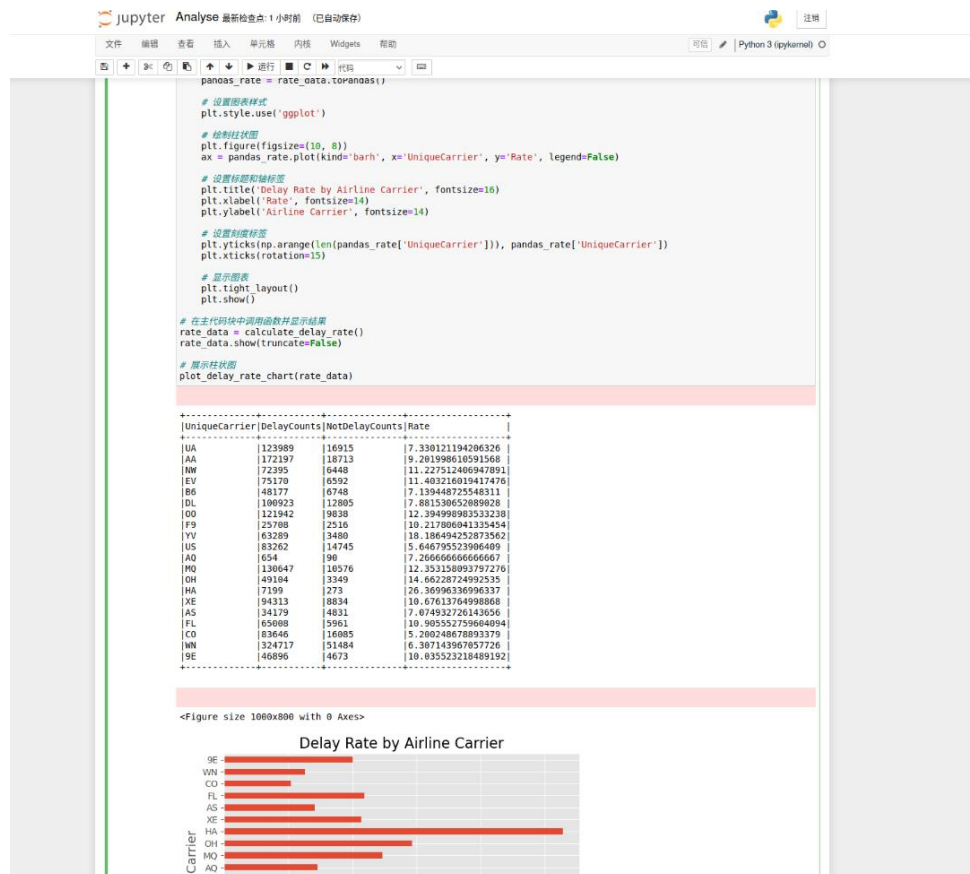
    # 设置刻度标签
    plt.yticks(np.arange(len(pandas_rate['UniqueCarrier'])),
pandas_rate['UniqueCarrier'])
    plt.xticks(rotation=15)

    # 显示图表
    plt.tight_layout()
    plt.show()

```

```
# 在主代码块中调用函数并显示结果
rate_data = calculate_delay_rate()
rate_data.show(truncate=False)
# 展示柱状图
plot_delay_rate_chart(rate_data)
```

运行过程如图：



8.4 分析一天中延误最严重的飞行时间

为了识别一天中航班延误最严重的时段，我们采用了以“小时”为单位的时间划分方法，将一天划分为 24 个独立的时间段，并计算了每个小时内航班的延误次数。在此分析中，我们关注了两个核心变量：航班的起飞时间（DepTime）和航班的抵达延误时间（ArrDelay）。

航班的起飞时间（DepTime）标志着航班计划的启动，它与航空公司的调度能力和机场的运营效率紧密相关，并且能够反映出航路积压对航班时间的影响。因此，我们依据起飞时间来划分时间段进行分析。同时，航班的抵达延误时间（ArrDelay）是判断航班是否延误的决定性指标。

考虑到 DepTime 是以数的形式存储的（格式为 hhmm），其在数据集中的表

现形式因时间段而异，例如凌晨 00:23 表示为 23.0，早上 7:45 表示为 745.0，而下午 15:13 则表示为 1513.0。为了从中提取准确的小时信息，我们首先将起飞时间转换为字符串格式，并根据其位数进行解析，从而将起飞时间转换为对应的小时数。

完成这一转换后，我们按照小时对数据进行分组，并统计了每个小时内的总延误航班数。通过比较一天中各个时间段的延误航班数量，我们可以确定延误最为严重的时间段。具体来说，延误航班数最多的时间段即为延误最为严重的时段。以下是我们采用的代码实现：

代码 5：分析一天中延误最严重的飞行时间

```
from pyspark.sql.functions import col, substring, avg, sum
from pyspark.sql.functions import expr
import numpy as np

def analyze_delay_by_day():
    """
    Task3:分析一天中延误最严重的时间段
    """
    # 选择所需的列
    columns_needed = ['DepTime', 'ArrDelay']
    data_selected = data.select(columns_needed)
    # 清洗数据，移除非抵达延误的航班
    data_cleaned = data_selected.filter((col('ArrDelay').cast('int') >= 0))
    # 从 DepTime 中提取小时信息
    data_cleaned = data_cleaned.withColumn('DepTimeStr', expr("cast(DepTime as string)"))
    data_with_hour = data_cleaned.withColumn(
        'Hour',
        expr("CASE "
            "WHEN length(DepTime) = 5 THEN cast(substring(DepTime, 0, 1) AS INT)
            "
            "WHEN length(DepTime) = 6 THEN cast(substring(DepTime, 0, 2) AS INT)
            "
            "WHEN length(DepTime) < 5 THEN cast('0' AS INT) "
            "ELSE NULL END")
    )
    # 计算每个小时的总延误航班数
    sum_delay_by_hour = data_with_hour.filter(col('ArrDelay') > 0) \
        .groupBy('Hour') \
        .agg(count('*').alias('SumOfDelayInOneHour'))
```

```

# 结果
delay_list = sum_delay_by_hour.orderBy(col('Hour'))
sum_delay_by_hour.orderBy(col('Hour')).show(24)
return delay_list

def plot_delay_by_day(delays):
    """
    绘制延误时间图表 by day
    """
    pandas_data = delays.toPandas()
    plt.style.use('seaborn-poster')
    # 绘制柱状图
    plt.figure(figsize=(10, 8))
    ax = pandas_data.plot(kind='line', x='Hour', y='SumOfDelayInOneHour',
marker='o', linestyle='-', legend=False,color='#15aabf')

    # 设置标题和轴标签
    plt.title('Delay by Hour of the Day', fontsize=18)
    plt.xlabel('Hour', fontsize=16)
    plt.ylabel('Flight delays per hour', fontsize=16)

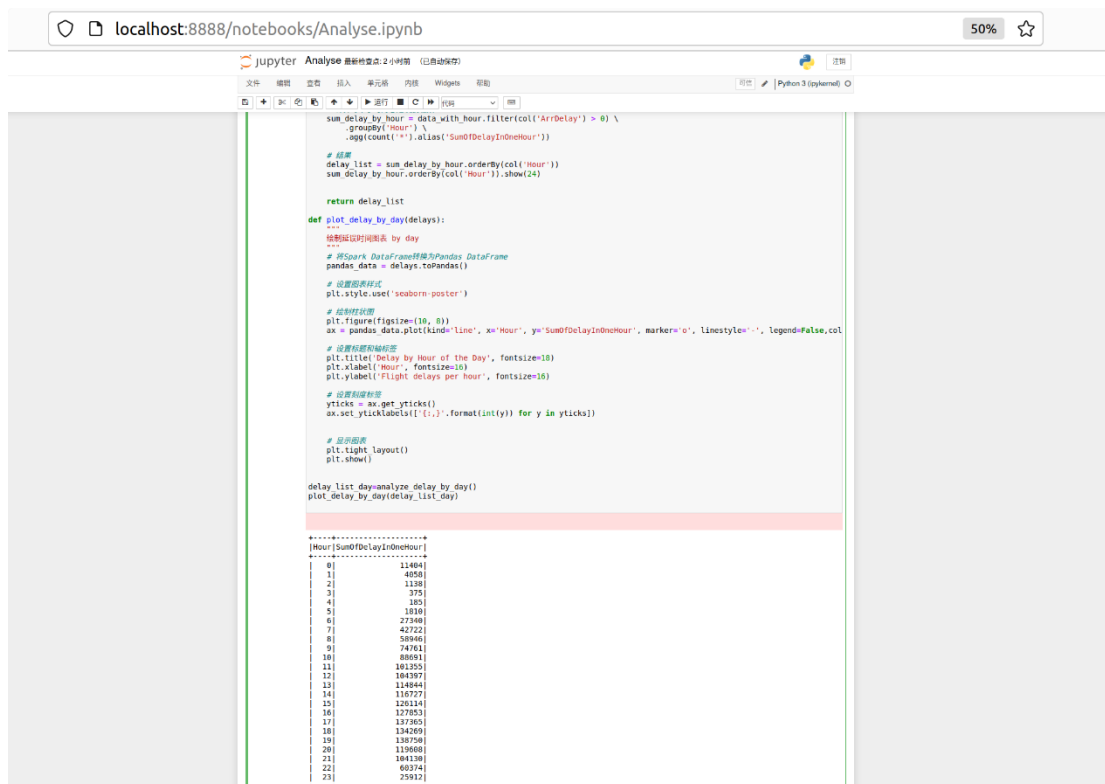
    # 设置刻度标签
    yticks = ax.get_yticks()
    ax.set_yticklabels(['{:,.}{}'.format(int(y), y) for y in yticks])

    # 显示图表
    plt.tight_layout()
    plt.show()

delay_list_day=analyze_delay_by_day()
plot_delay_by_day(delay_list_day)

```

运行结果如图



8.5 分析一周中延误最严重的飞行时间

本题与上一题相似，从一周的各日当中延误的航班架次总数考虑，延误架次最大的一日为延误最严重的日子。这里我们选取：星期几（DayOfWeek）和航班的抵达延误时间（ArrDelay）作为分析变量。

按照星期几（DayOfWeek）对数据进行分组，并统计了每个天内的总延误航班数。通过比较周中每天的延误航班数量，我们可以确定一周内容延误最为严重的时间日。总体代码逻辑与上一题相似，不再赘述。

代码如下所示——

代码 6：分析一周中延误最严重的飞行时间

```
from pyspark.sql.functions import col, substring, avg, sum
from pyspark.sql.functions import expr
import numpy as np

def analyze_delay_by_week():
    """
    Task4:分析一周中延误最严重的时间段
    """

    # 选择所需的列
    columns_needed = ['DayOfWeek', 'ArrDelay']
    data_selected = data.select(columns_needed)
```

```

# 清洗数据，移除非抵达延误的航班

data_cleaned = data_selected.filter((col('ArrDelay').cast('int') >= 0))
# 计算周中每天的总延误航班数
sum_delay_by_week = data_cleaned.filter(col('ArrDelay') > 0) \
    .groupBy('DayOfWeek') \
    .agg(count('*').alias('SumOfDelayInOneDay'))

# 结果
delay_list = sum_delay_by_week.orderBy(col('DayOfWeek'))
sum_delay_by_week.orderBy(col('DayOfWeek')).show(7)
return delay_list

def plot_delay_by_week(delays):
    """
    绘制延误时间图表 by week
    """
    pandas_data = delays.toPandas()
    plt.style.use('seaborn-poster')

    # 设置星期映射
    day_mapping = {1: 'Mon', 2: 'Tue', 3: 'Wed', 4: 'Thu', 5: 'Fri', 6: 'Sat', 7:
'Sun'}
    pandas_data['DayOfWeek'] = pandas_data['DayOfWeek'].map(day_mapping)

    # 绘制柱状图
    plt.figure(figsize=(10, 8))
    ax = pandas_data.plot(kind='line', x='DayOfWeek', y='SumOfDelayInOneDay',
marker='o', linestyle='-', legend=False,color='#7950f2')

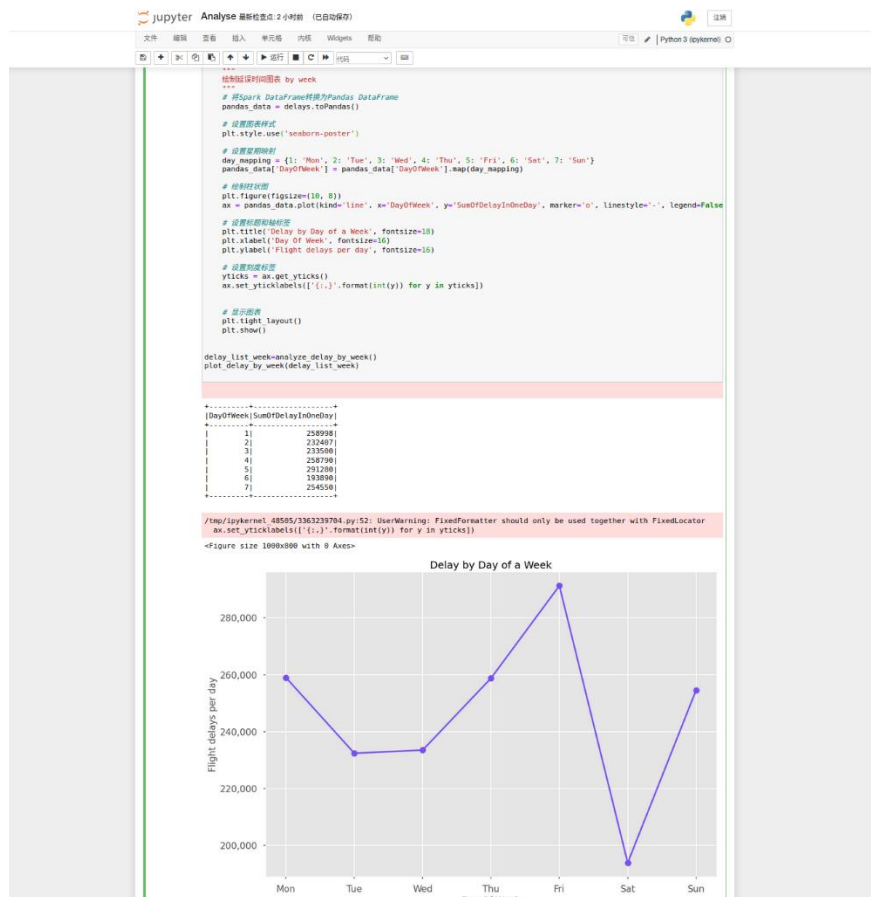
    # 设置标题和轴标签
    plt.title('Delay by Day of a Week', fontsize=18)
    plt.xlabel('Day Of Week', fontsize=16)
    plt.ylabel('Flight delays per day', fontsize=16)
    yticks = ax.get_yticks()
    ax.set_yticklabels(['{:,}'.format(int(y)) for y in yticks])

    # 显示图表
    plt.tight_layout()
    plt.show()

delay_list_week=analyze_delay_by_week()
plot_delay_by_week(delay_list_week)

```

结果如图，具体分析见第九部分。查阅资料可知，出现的相关警告可忽略。



8.6 短途航班和长途航班，哪种航班取消更严重？

根据《通用航空短途运输运营服务管理办法》的规定，短途飞行通常指的是那些跨省、自治区、直辖市（统称为省）的飞行，其航程一般不超过 500 公里。在本项研究中，为了简化分析，我们将不考虑航班的跨省/州特性，仅以航程长度作为划分短途与长途航班的唯一标准。因此，我们将通过对比航程（Distance）字段与 500 公里（约合 310.7 英里）这一阈值来判定航班是属于短途还是长途。

本研究的核心任务是对比分析短途与长途航班中取消航班的相对比例。具体而言，我们将关注那些在 Cancelled 字段中标记为 1 的航班，并计算这些取消航班在短途和长途航班总数中的占比（取消率）。为此，我们将采用 Distance 和 Cancelled 这两个关键指标来进行数据分析。代码如下所示

代码 7: 分析短途航班和长途航班哪种航班取消更严重

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, count, round as bround
import numpy as np

def cancelRateByFlightType():
```

```

"""
Task5: 计算短途航班和长途航班的取消率
"""

# 选取问题所需特征
flight_data = data.select('Distance', 'Cancelled')

# 标记短途和长途航班
flight_data = flight_data.withColumn('FlightType', when(col('Distance') <=
310.7, 'Short').otherwise('Long'))

# 统计取消的航班数目
cancelled_flights = flight_data.where(col('Cancelled') ==
1).groupBy('FlightType').count()
cancelled_flights = cancelled_flights.withColumnRenamed('count', 'CancelNum')

# 统计未取消的航班数目
non_cancelled_flights = flight_data.where(col('Cancelled') ==
0).groupBy('FlightType').count()
non_cancelled_flights = non_cancelled_flights.withColumnRenamed('count',
'NoCancelNum')

# 合并取消和未取消的航班数目
merged_data = cancelled_flights.join(non_cancelled_flights, on='FlightType',
how='left_outer')

# 计算总航班数
merged_data = merged_data.withColumn('Total', col('CancelNum') +
col('NoCancelNum'))

# 计算取消率
merged_data = merged_data.withColumn('CancelRate', round(col('CancelNum') /
col('Total'), scale=6))
merged_data.show()
return merged_data

def plotCancelRateByFlightType(cancel_rate_data):
    """
    绘制短途和长途航班的取消率
    """

    pandas_cancel_rate = cancel_rate_data.toPandas()
    plt.style.use('ggplot')

    # 创建条形图
    fig, ax = plt.subplots(figsize=(10, 6))
    bar_colors = ['skyblue', 'salmon']
    bar_width = 0.5
    index = np.arange(len(pandas_cancel_rate['FlightType']))
    bars = ax.barh(index, pandas_cancel_rate['CancelRate'], bar_width,
color=bar_colors)

    # 设置标题和标签
    ax.set_xlabel('Cancel Rate', fontsize=14)

```

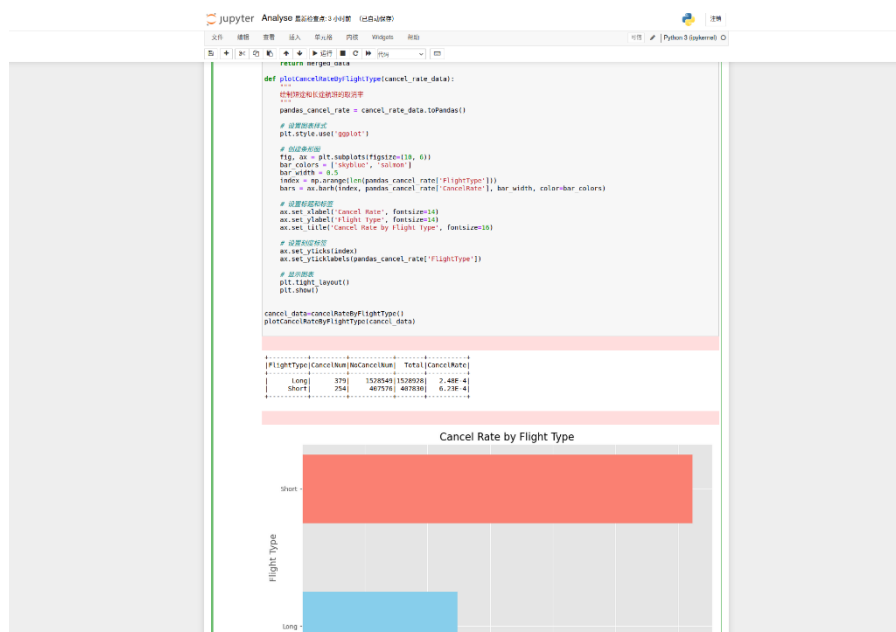
```

ax.set_ylabel('Flight Type', fontsize=14)
ax.set_title('Cancel Rate by Flight Type', fontsize=16)
ax.set_yticks(index)
ax.set_yticklabels(pandas_cancel_rate['FlightType'])
plt.tight_layout()
plt.show()

cancel_data=cancelRateByFlightType()
plotCancelRateByFlightType(cancel_data)

```

运行过程如图，详细分析见第九部分。



8.7 建立机器学习算法模型，预测未来航班取消情况

在这里，我们使用支持向量机（SVM）、随机森林（Random Forest）、梯度提升树(GBDT)这几种典型机器学习算法进行预测。

（1）预处理航班数据集

我们读取了名为DelayedFlights.csv的CSV文件，并将其存储在变量flightData中。然后，代码定义了可预测的特征和目标变量，其中可预测特征包括年份、月份、日、星期、计划起飞时间、计划到达时间、唯一承运人、航班号、机尾号、计划飞行时间和距离等，而目标变量为Cancelled（是否取消）。

接下来，代码从数据集中选择了特征和目标变量，并删除了含有缺失值的行，以确保特征矩阵和目标变量索引的一致性。随后，代码定义了哪些特征是分类特征，哪些是数值特征，并创建了一个ColumnTransformer来分别对这两类特征进

行处理。数值特征通过 `StandardScaler` 进行标准化，而分类特征则通过 `OneHotEncoder` 进行独热编码。

在预处理完成后，代码将处理后的特征矩阵应用于训练集和测试集的分割，其中测试集占总数据的 20%，并设置了随机种子以确保结果的可重复性。至此，数据预处理和分割工作已完成，为后续的模型训练和预测做好了准备。

具体代码如下——

代码 8：数据预处理与训练集划分

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split

# 读取数据
flightData = pd.read_csv('./DelayedFlights.csv')
data=flightData
data.head(10)

# 定义可预知的特征和目标变量
predictable_features = ['Year', 'Month', 'DayofMonth', 'DayOfWeek', 'CRSDepTime',
                        'CRSArrTime', 'UniqueCarrier', 'FlightNum', 'TailNum', 'CRSElapsedTime',
                        'Distance', 'Origin', 'Dest']
target_variable = 'Cancelled'

# 选择特征和目标变量
X = data[predictable_features]
y = data[target_variable]

# 删除含有缺失值的行
X = X.dropna()
y = y[X.index] # 确保 y 与 X 的索引对齐

# 定义分类特征和数值特征
categorical_features = ['UniqueCarrier', 'FlightNum', 'TailNum', 'Origin',
                        'Dest']
numerical_features = ['Year', 'Month', 'DayofMonth', 'DayOfWeek', 'CRSDepTime',
                      'CRSArrTime', 'CRSElapsedTime', 'Distance']

# 创建 ColumnTransformer 来分别处理分类和数值特征
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
```



```

        ('cat', OneHotEncoder(), categorical_features)

    ])

# 应用预处理
X_processed = preprocessor.fit_transform(X)

# 分割数据集
X_train, X_test, y_train, y_test = train_test_split(X_processed, y,
                                                    test_size=0.2, random_state=42)

```

```

### 数据处理

1 数据处理

In [1]: import pandas as pd
        from sklearn.preprocessing import StandardScaler, OneHotEncoder
        from sklearn.compose import ColumnTransformer
        from sklearn.model_selection import train_test_split

        flightData = pd.read_csv("../DelayedFlights.csv")
        data = flightData

In [2]: # 统计 Canceled 中值为 1 的个数
        cancel_count = data[data['Canceled'] == 1].shape[0]
        print("Number of 'cancel' entries with value 1: (cancel count)")

        Number of 'cancel' entries with value 1: 633

In [3]: data.head(10)

Out[3]:
   Unnamed: 0  Year  Month  DayofMonth  DayOfWeek  DepTime  CRSDepTime  ArrTime  CRSArrTime  UniqueCarrier  TailNum  TaxiOut  Canceled  Cancelled
0           0    2008     1         3         4      2003.0      1955.0  2781.0      2775.0      WN          400     8.0      0.0      0
1           1    2008     1         3         4       714.0       715.0  1007.0      1000.0      WN          330    10.0      0.0      0
2           2    2008     1         3         4       658.0       655.0   904.0       790.0      WN          330    11.0      0.0      0
3           3     2008     1         3         4      1020.0      1755.0  1959.0      1925.0      WN          330    10.0      0.0      0
4           4     2008     1         3         4      1040.0      1915.0  2121.0      2190.0      WN          400    10.0      0.0      0
5           5     2008     1         3         4      1037.0      1830.0  2037.0      1940.0      WN          330     7.0      0.0      0
6           6    10  2008     1         3         4       700.0       700.0   916.0       915.0      WN          500    10.0      0.0      0
7           7    10  2008     1         3         4      1044.0      1310.0  1440.0      1775.0      WN          500     9.0      0.0      0
8           8    10  2008     1         3         4      1019.0      1330.0  1531.0      1690.0      WN          500     9.0      0.0      0
9           9    10  2008     1         3         4      1412.0      1420.0  1640.0      1625.0      WN          700     8.0      0.0      0

10 rows x 14 columns

In [4]: # 定义可预测特征和不可预测特征
        predictable_features = ['Year', 'Month', 'DayofMonth', 'DayOfWeek', 'CRSDepTime', 'CRSArrTime', 'UniqueCarrier', 'FlightNum', 'TailNum', 'CRSElaps']
        target_variable = 'Canceled'

        # 分割数据集
        X = data[predictable_features]
        y = data[target_variable]

        # 训练模型

```

（2）SVM 支持向量机

下面，我们用 SVM 支持向量机进行训练、预测。

首先，我们计算了类权重，这是因为我们的数据集中航班取消的条目较少，因此我们使用 **balanced** 模式来为每个类别分配权重。**balanced** 模式是一种处理不平衡数据集的方法，它通过调整每个类别的权重，使得模型在训练过程中能够更加关注那些样本数量较少的类别。这样做可以防止模型仅仅因为多数类样本多而偏向于预测多数类，从而提高模型对少数类的识别能力，确保模型不会偏向于多数类。

接着，我们初始化了一个 **SVM** 分类器，并选择了线性核，同时应用了计算得到的类权重，并设置了随机状态以确保结果的可重复性。然后，我们使用训练集来训练模型。模型训练完成后，我们用测试集进行预测，并计算了准确率、分类报告（包括精确率、召回率和 F1 分数）。接着，我们计算了 ROC 曲线和 AUC 值以评估模型性能，并打印了召回率和 AUC 值，同时绘制了 ROC 曲线来直观评估模型在预测航班取消方面的表现。

相关代码如下：

代码 9：SVM 支持向量机训练与评测

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# ====开始训练模型=====
# 计算类权重
class_weights = compute_class_weight(class_weight='balanced',
                                     classes=np.unique(y_train), y=y_train.values)
class_weights_dict = dict(enumerate(class_weights))

# 使用 SVM 分类器，并设置类权重
svm_clf = SVC(kernel='linear', class_weight=class_weights_dict, random_state=42)
# 使用线性核

# 训练模型
svm_clf.fit(X_train, y_train.values)

# ====开始测试模型=====
# 预测测试集
y_pred = svm_clf.predict(X_test)

# 评估模型性能
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
# 打印准确率和分类报告
print(f"Accuracy: {accuracy}")
print(f"Classification Report:\n{classification_rep}")
from sklearn.metrics import accuracy_score, classification_report, roc_curve,
auc, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
from scipy import interp
from itertools import cycle

# Binarize the output
y_test_bin = label_binarize(y_test, classes=[0, 1])
y_pred_bin = label_binarize(y_pred, classes=[0, 1])

# 计算 ROC 曲线和 AUC
fpr, tpr, _ = roc_curve(y_test_bin.ravel(), y_pred_bin.ravel())
```

```

roc_auc = auc(fpr, tpr)
# 打印召回率和AUC
print(f"Recall: {tpr[1]}")
print(f"AUC: {roc_auc}")
# 绘制ROC 曲线
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('SVM Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

```

The screenshot shows a Jupyter Notebook interface with the following code and output:

```

In [11]: from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# 计算权重
class_weight = compute_class_weight('balanced', classes=np.unique(y_train), y_train=y_train)
class_weight_dict = dict(zip(np.unique(y_train), class_weight))

# 使用SVM分类器
svm_clf = SVC(kernel='linear', class_weight=class_weight_dict, random_state=42) # 使用过拟合

In [12]: svm_clf.fit(X_train, y_train.values)

Out[12]: SVC(class_weight={0: 0.5001596063955836, 1: 1.9448929292929292},
kernel='linear', random_state=42)

In [13]: # 预测测试集
y_pred = svm_clf.predict(X_test)

# 计算准确率
accuracy = accuracy_score(y_test, y_pred)
classification_report = classification_report(y_test, y_pred)

# 打印准确率
print('Accuracy (accuracy):')
print(classification_report(classification_report))

Accuracy: 0.998248901999427
Classification Report
precision recall f1 score support
0 1.00 1.00 1.00 387173
1 0.02 0.07 0.03 138
accuracy 1.00 387311
macro avg 0.51 0.53 0.51 387311
weighted avg 1.00 1.00 1.00 387311

In [14]: from sklearn.metrics import accuracy_score, classification_report, roc_curve, auc, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

# 二值化输出
y_test_bin = label_binarize(y_test, classes=(0, 1))
y_pred_bin = label_binarize(y_pred, classes=(0, 1))

```

(3) 随机森林

首先，导入必要的库和模块（包括随机森林分类器、管道、评估指标等）来准备环境。接着，我们重新定义了一个预处理管道，该管道包括数值特征的标准化处理 and 类别特征的一键编码处理。然后，创建了一个随机森林分类器，并设置了类别权重为平衡，以处理数据不平衡的问题。之后，将预处理步骤和分类器组合成一个管道，并使用训练集数据训练随机森林模型。在模型训练完成后，使用测试集数据进行预测，并计算准确率、分类报告以及混淆矩阵来评估模型性能——计算了 ROC 曲线和 AUC 值。最后，绘制了 ROC 曲线图，展示了模型的召回率和 AUC 值，从而直观地评估了模型的预测能力。

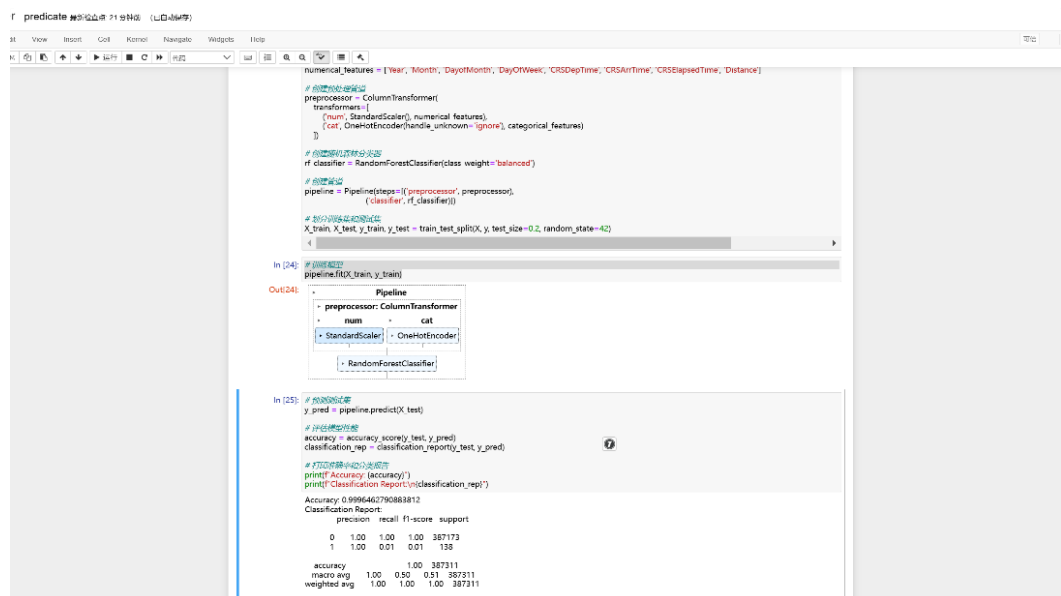
具体代码如下——

代码 10：随机森林训练与评测

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score
import numpy as np

# ==== 开始训练模型=====
# 创建预处理管道
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])
# 创建随机森林分类器
rf_classifier = RandomForestClassifier(class_weight='balanced')
# 创建管道
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', rf_classifier)])
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)
# 训练模型
pipeline.fit(X_train, y_train)

# ==== 开始测试模型=====
# 预测测试集
y_pred = pipeline.predict(X_test)
# 剩余测试代码与前叙模型的评测代码基本一致，故略...
```



```
# In [24]: pipeline.fit(X_train, y_train)
Out[24]: Pipeline
- preprocessor: ColumnTransformer
- num - cat
- StandardScaler - OneHotEncoder
- RandomForestClassifier

# In [25]: # 开始测试
y_pred = pipeline.predict(X_test)
# 计算准确率
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
# 打印结果
print('Accuracy (accuracy)')
print('Classification Report:')
print(classification_rep)
Accuracy: 0.9996462790883812
Classification Report:
precision recall f1-score support
0 1.00 1.00 1.00 387173
1 1.00 0.01 0.01 138

accuracy
macro avg 1.00 0.50 0.51 387311
weighted avg 1.00 1.00 1.00 387311
```

(4) 梯度提升树(GBDT)

最后，使用梯度提升树（Gradient Boosting Decision Tree, GBDT）算法来预测航班是否取消。代码逻辑如下：首先，然后创建一个预处理管道，该管道包括数值特征的标准化和类别特征的一键编码。随后，计算样本权重以处理数据不平衡问题。

紧接着，我们运用“早停策略”来推进训练循环。早停策略通过设定一个“耐心值”来预防过拟合现象，这在处理不平衡数据集（如本数据集中取消航班的样本数量相对较少）时尤为重要。在这种数据分布下，模型有可能会偏向于多数类，而忽略对少数类的学习。长时间的训练可能导致模型在多数类上过度拟合。早停策略通过监测验证集上的性能，一旦发现性能不再提升，便立即停止，从而避免了对多数类的过拟合。最后，对测试集进行预测，评估模型性能。代码如下——

代码 11：梯度提升树 GBDT 训练与评测

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, roc_curve, auc
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.utils.class_weight import compute_sample_weight
import numpy as np

# ====开始训练模型=====
# 创建预处理管道
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])

# 划分训练集和验证集
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
    random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,
    test_size=0.25, random_state=42) # 0.25 x 0.8 = 0.2

# 创建梯度提升树分类器
```

```

gbdt_classifier = GradientBoostingClassifier(n_estimators=1000,
learning_rate=0.1, max_depth=3, random_state=42, warm_start=True)
# 创建管道
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                             ('classifier', gbdt_classifier)])

# 计算样本权重
sample_weights = compute_sample_weight(class_weight='balanced', y=y_train)

# 初始化验证集的最佳性能和迭代次数
best_val_score = 0
best_iter = 0
patience = 10
patience_counter = 0

# 训练循环，实现早停
for i in range(1, 1001):
    pipeline.named_steps['classifier'].n_estimators = i
    pipeline.fit(X_train, y_train, classifier__sample_weight=sample_weights)

    # 验证集性能
    val_predictions = pipeline.predict(X_val)
    val_score = accuracy_score(y_val, val_predictions)

    # 如果验证集性能提升，更新最佳性能和迭代次数
    if val_score > best_val_score:
        best_val_score = val_score
        best_iter = i
        patience_counter = 0
    else:
        patience_counter += 1

    # 如果耐心耗尽，停止训练
    if patience_counter >= patience:
        print(f"Early stopping at iteration {i}, best validation score:
{best_val_score}")
        break

# 使用最佳迭代次数更新模型
pipeline.named_steps['classifier'].n_estimators = best_iter

# ====开始测试模型=====
# 预测测试集
y_pred = pipeline.predict(X_test)
# 剩余测试代码与前叙模型的评测代码基本一致，故略...

```

```
predicate 训练地点: 37 分钟前 (默认未保存)
View Insert Cell Kernel Navigate Widgets Help 可运行

else:
    patience_counter += 1
    # 训练核心模型, 停止训练
    if patience_counter == patience:
        print("Early stopping at iteration (%i, best validation score: (%best_val_score))")
        break
    # 使用最佳模型进行预测
    pipeline.named_steps[classifier].n_estimators = best_n_estimators
    Early stopping at iteration 12, best validation score: 0.8286725654577329

In [29]: # 开始测试模型
y_pred = pipeline.predict(X_test)

# 评估模型性能
accuracy = accuracy_score(y_test, y_pred)
classification_report = classification_report(y_test, y_pred)

# 打印准确率和分类报告
print("Accuracy: (%accuracy)")
print("Classification Report: (%classification_report)")

Accuracy: 0.8286725654577329
Classification Report:
precision recall f1-score support
0 1.00 0.83 0.91 387181
1 0.00 0.98 0.00 130
accuracy 0.83 387311
macro avg 0.50 0.91 0.45 387311
weighted avg 1.00 0.83 0.91 387311

In [41]: # 二值化输出
y_test_bin = label_binarize(y_test, classes=[0, 1])
y_pred_bin = label_binarize(y_pred, classes=[0, 1])

# 计算ROC曲线下面积
fpr, tpr, _ = roc_curve(y_test_bin, y_pred_bin)
roc_auc = auc(fpr, tpr)
```

至此，我们共完成了支持向量机、随机森林、梯度提升树三种典型模型的训练与预测，具体的分析结果和评测指标将在第九部分详细阐述。

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。

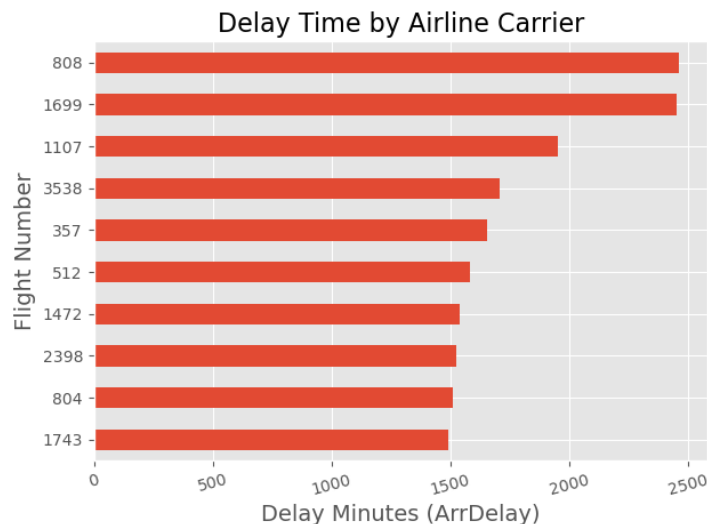
以“抵达延误时间”ArrDelay 作为衡量标准进行数据分析。得到上图所示的结果。列表展示为数据集中前 10 名延误时间最长的航班（已按延误时间长短排序），第一列为航班号，第二列为延误时间。

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

可以看到延误最长的航班整整延迟了 2461 分钟，该航班为 808 号航班。紧随其后的是 1699 号航班，延误时间为 2453 分钟。

为了更清晰描述这个数据，我们可以用一个柱状图可视化展现，如下图所示。横轴代表延误的分钟数，纵轴是航班号编码。



2. 计算延误的和没有延误的航空公司的比例。

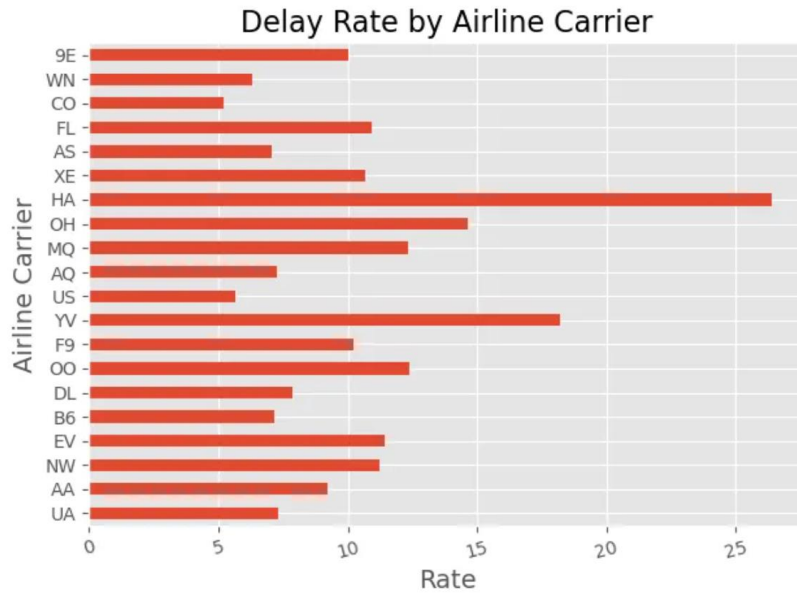
```
Number of airlines with delay: 20
Number of airlines without delay: 0
The proportion of delayed to non-delayed airlines: inf
```

根据我们的代码运行，没有完全不延误的航司。即所有航司都或多或少有航班存在延误状态。于是，我们统计各航空公司延误与未延误方面的航班比例。

下图展示了各航空公司延误与未延误的航班情况比例列表。第一列为航空公司编码，第二列为延误航班数（DelayCounts），第三列为未延误航班数（NotDelayCounts），最后一列为该航司对应的延误与未延误比例。

UniqueCarrier	DelayCounts	NotDelayCounts	Rate
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
US	83262	14745	5.646795523906409
AQ	654	90	7.266666666666667
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
FL	65008	5961	10.905552759604094
CO	83646	16085	5.200248678893379
WN	324717	51484	6.307143967057726
9E	46896	4673	10.035523218489192

从表格中可以看出，不同航空公司的航班延误和未延误情况存在显著差异。可以看出 HA 航空公司延误比最高，达到了 26.36；CO 航空公司最低，仅 5.2。我们还可以用图形化表示，更直观反应各航司的延误比。



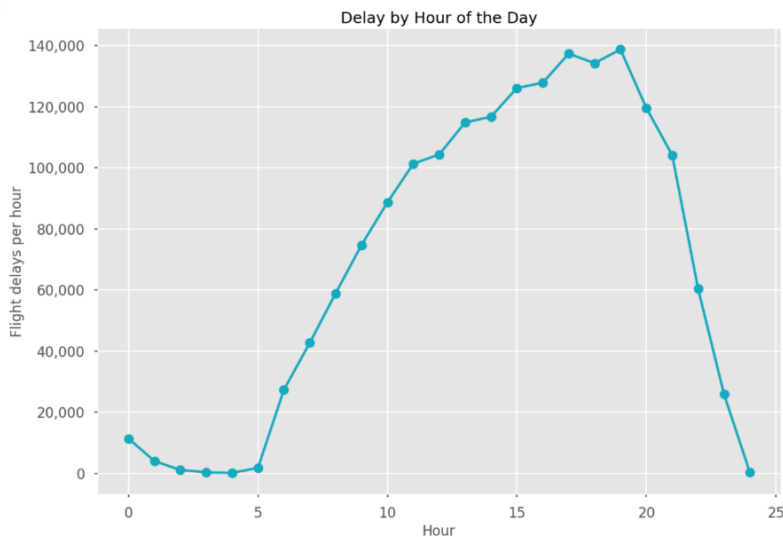
3. 分析一天中延误最严重的飞行时间。

Hour	SumOfDelayInOneHour
0	11404
1	4058
2	1138
3	375
4	185
5	1810
6	27340
7	42722
8	58946
9	74761
10	88691
11	101355
12	104397
13	114844
14	116727
15	126114
16	127853
17	137365
18	134269
19	138750
20	119608
21	104130
22	60374
23	25912

only showing top 24 rows

为了分析一天中延误最严重的飞行时间，我们将一天按每小时划分为 24 个时间段（以起飞时间作为时段划分依据）。统计各个时间段航班延误频次（架次）。如上图所示，第一列代表时间段（0~23 小时），第二列为该时段对应的延误频次。

为了更清晰的反应一天中各个时段的延误情况，我们绘制了折线图以更直观的反应。



我们可以明显观察到，在凌晨时段（0 时至 5 时），航班的延误情况相对较少。这主要是因为在这一时间段，大多数人正处于休息状态，导致航班数量减少，航路较为畅通，因此延误事件的发生率较低；随着清晨的到来，白天的活动逐渐展开，航班的延误情况也开始逐渐增加。特别是在上午 11 点至晚上 21 点这一时间段，延误的航班数量激增至 100,000 班次。这一现象可能与乘客更倾向于在这个时间段出行，导致机场客流量增大，航路出现积压有关。随后，夜幕降临，乘客的数量逐渐减少，导致航班的飞行次数也随之降低。因此，航班的延误率也逐渐下降，恢复了更为平稳的运行状态。

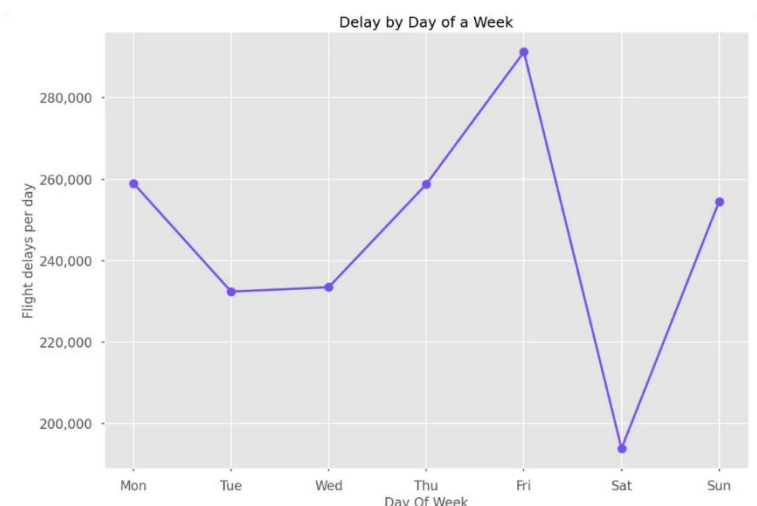
延误情况的顶峰出现在傍晚 17 时至 19 时，这一时段被认为是“晚高峰”阶段，延误航班数高达 138,750 班次。许多乘客选择在这个时间段进行回程旅行或商务出行，进一步加剧了航班的延误情况。**因此该时段（17 时~19 时）可以认为是一天中延误最严重的飞行时间。**

4. 分析一周中延误最严重的飞行时间。

与上一个问题同理，我们按“星期几”划分一周的时间段，以裁定一周中延误最严重的飞行时间。下表展示了周一到周日每天的航班延误架次数量。

DayOfWeek	SumOfDelayInOneDay
1	258998
2	232407
3	233500
4	258790
5	291280
6	193890
7	254550

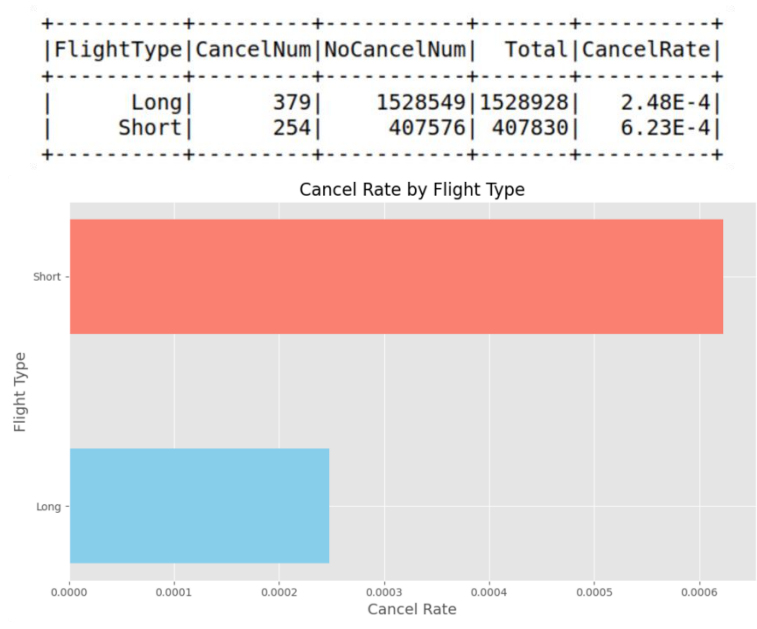
同时，我们绘制了相应的折线图



这张图显示了每周不同日的航班延误情况。从图中可以看出，周五的航班延误次数最高，延误架次高达 291280 架次，而周六的延误次数最低。因为周五通常是工作周的最后一天，旅行乘客、出差返程乘客较多，航班数量较多，机场和空管系统面临更大的压力，导致延误增加。而周六则相对较少，可能是因为周末的航班安排较为宽松。因此，周五是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？

根据相关规定，我们将航程超过 500 公里（310.7 英里）的视为长途航班，分类统计每类航班的取消率，可以得到下表和下图的结果——



从图表可以看出，短途航班的取消率约为 0.0623%，远高于长途航班的取消率 0.0248%。短途航班的取消率约为长途的三倍。可能的原因是短途航班可能面临更大的经济压力，因为它们的票价通常低于长途航班。这可能导致航司在运营决策上更加谨慎，例如在特殊的情况下倾向选择取消短途航班，保障长途航班。因此，短途航班取消率更高，短途航班取消情况更严重。

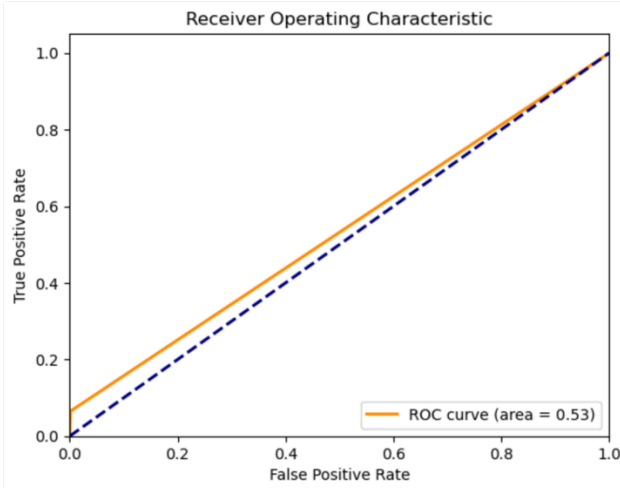
6. 建立机器学习算法模型，预测未来航班取消情况。

我们使用了支持向量机（SVM）、随机森林（Random Forest）、梯度提升树（GBDT）这三种典型机器学习算法进行预测，下面探究他们的预测情况。

(1) SVM

Accuracy: 0.9982649601999427				
Classification Report:				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	387173
1	0.02	0.07	0.03	138
accuracy			1.00	387311
macro avg	0.51	0.53	0.51	387311
weighted avg	1.00	1.00	1.00	387311

上图给出了 SVM 的预测情况。虽然整体准确率高达 99.83%，看似表现极佳，但实际上对取消航班的预测能力有限。模型在预测未取消航班（类别 0）时表现出完美的精确率和召回率，但对取消航班（类别 1）的预测则显得无力，精确率仅为 2%，召回率仅为 7%，这是由于数据集中取消航班的样本较少造成的。



AUC 值仅为 0.5319，接近 0.5，模型在预测取消航班任务上的效果一般。

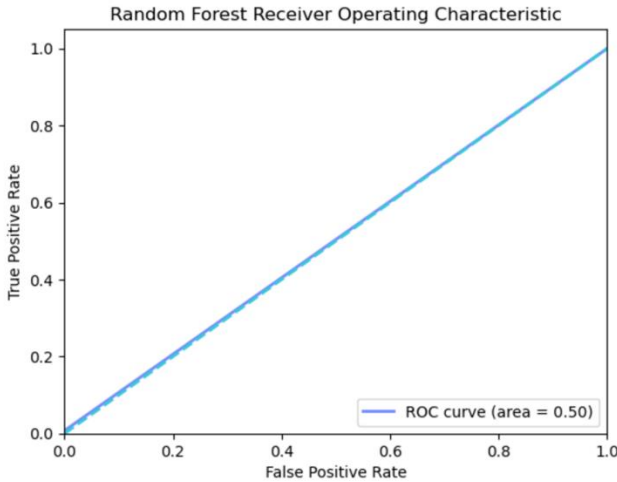
(2) 随机森林

下图给出了随机森林的预测情况。随机森林模型在预测航班取消情况时表现出很高的准确率（0.9996），这意味着模型在大多数情况下都能正确预测航班是否会取消。然而，模型在处理正类（航班取消）的预测上存在明显不足。

Accuracy: 0.9996462790883812				
Classification Report:				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	387173
1	1.00	0.01	0.01	138
accuracy			1.00	387311
macro avg	1.00	0.50	0.51	387311
weighted avg	1.00	1.00	1.00	387311

具体来说，模型的召回率仅为 0.0072，这表明模型在所有实际取消的航班中，只识别出了极少数。虽然精确度为 1.00，但由于正类样本数量极少，这个指标并不能很好地反映模型性能。

Recall: 0.007246376811594203
AUC: 0.5036231884057971



AUC 值为 0.5036，接近 0.5，随机森林模型在预测航班取消情况时存在明显的偏向，需要进一步优化以提高对正类样本的识别能力。

(3) 梯度提升树

Accuracy: 0.8284763407184408
Classification Report:

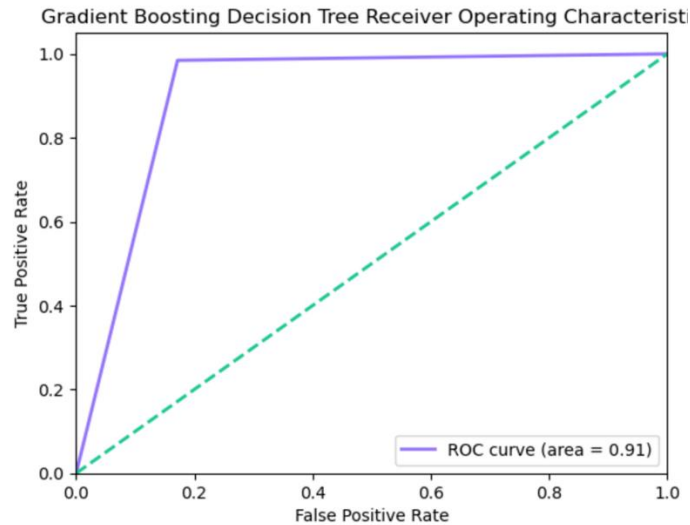
	precision	recall	f1-score	support
0	1.00	0.83	0.91	387181
1	0.00	0.98	0.00	130

accuracy			0.83	387311
macro avg	0.50	0.91	0.45	387311
weighted avg	1.00	0.83	0.91	387311

梯度提升树模型在预测航班取消情况时表现出了较高的准确性和有效性。模型的准确率为 0.828，这意味着模型正确预测了约 83% 的航班取消情况。在分类报告中，对于未取消的航班（标签为 0），模型的精确度为 1.00，召回率为 0.83，F1 分数为 0.91，这表明模型在预测未取消航班方面表现非常好。对于取消的航班（标签为 1），召回率高达 0.98，说明模型在识别取消航班方面几乎不会漏掉任何真实取消的航班。

在对比 SVM 和随机森林模型时，梯度提升树模型的性能更优。如下图所示，梯度提升树的 AUC 值为 0.907，说明模型在区分取消与非取消航班方面的能力较强，这优于 SVM 和随机森林模型。虽然模型在精确度方面存在一定的不平衡，但整体来看，梯度提升树模型在预测航班取消情况时，相较于 SVM 和随机森林模型，具有更好的效果。

Recall: 0.9846153846153847
AUC: 0.9065196500225594



十一、总结及心得体会

在本次针对航空公司延误和取消情况的实验分析中，小组同学展现出了高效且有序的合作态势。通过集体讨论，明确了任务分工，确保了实验的有序推进。在合作中，我们共同应对挑战，通过不断优化数据汇总流程，显著提升了报告的整体质量。

小组同学能够熟练运用大数据分析工具，能够基于 PySpark 进行数据处理。借助 Python 编程，我们顺利实现了数据的采集、处理、分析与可视化。在实验的推进过程中，通过团队成员间的互助与交流，以及网络学习新知识，我们逐一解决了诸多细节问题。这不仅巩固了我们在理论课程中学到的知识，也加深了我们对大数据分析实践的理解。

本次实验的一个显著特点是理论与实践的紧密结合。虽然我们在理论课程中学习了大数据计算系统的数据模型和处理算法，但实际操作中遇到的挑战让我们意识到理论与实践之间的差距。

通过这次实验，我们不仅锻炼了独立思考和团队协作的能力，还对大数据分析与计算领域有了更深入的认识。

十二、对本项目过程及方法、手段的改进建议

本实验所给出的数据集中，航班取消的样本数量较少（航班取消与航班未取消样本不均衡）。因此，在建立机器学习算法模型，预测未来航班取消情况时需要进一步考虑不平衡数据集的影响。