#data\_clean.py

from pyspark.sql import SparkSession

if **name** == '**main**': spark = SparkSession.builder.   
appName("data\_clean").   
master("yarn").   
config("spark.sql.shuffle.partitions", 2).   
config("spark.sql.warehouse.dir", "hdfs://node1:8020/user/hive/warehouse").   
config("hive.metastore.uris", "thrift://node1:9083").   
enableHiveSupport().   
getOrCreate() sc = spark.sparkContext

# 删除函数

def clean(df):

# 删除包含任何缺失值的行

df\_filled = df.na.drop()

# 删除重复行

df\_unique = df\_filled.dropDuplicates()

return df\_unique

#对4个表处理

df\_courier\_device = clean(spark.table("myhive.courier\_device"))

df\_courier\_device.write.mode("overwrite").saveAsTable("myhive.courier\_device\_clean")

df\_merchant\_device = clean(spark.table("myhive.merchant\_device"))

df\_merchant\_device.write.mode("overwrite").saveAsTable("myhive.merchant\_device\_clean")

df\_test\_courier\_feedback = clean(spark.table("myhive.courier\_feedback"))

df\_courier\_feedback = df\_test\_courier\_feedback.where("acceptance\_hour >= 0 AND acceptance\_minute >= 0 AND "

"acceptance\_second >= 0 AND feedback\_hour >= 0 AND "

"feedback\_minute >= 0 AND feedback\_second >= 0 AND feedback "

">= 0 AND distance\_to\_merchant >= 0 AND "

"distance\_to\_merchant <= 1000")

df\_courier\_feedback.write.mode("overwrite").saveAsTable("myhive.courier\_feedback\_clean")

df\_test\_courier\_report = clean(spark.table("myhive.courier\_report"))

df\_courier\_report = df\_test\_courier\_report.where("acceptance\_hour >= 0 AND acceptance\_minute >= 0 AND "

"acceptance\_second >= 0 AND arrier\_hour >= 0 AND arrier\_minute "

">= 0 AND arrier\_second >= 0 AND pickup\_hour >= 0 AND "

"pickup\_minute >= 0 AND pickup\_second >= 0 AND delivery\_hour >= "

"0 AND delivery\_minute >= 0 AND delivery\_second >= 0")

df\_courier\_report.write.mode("overwrite").saveAsTable("myhive.courier\_report\_clean")

#查看删除情况

spark.table("myhive.courier\_device").show()

spark.table("myhive.courier\_device\_clean").show()

spark.table("myhive.merchant\_device").show()

spark.table("myhive.merchant\_device\_clean").show()

spark.table("myhive.courier\_feedback").show()

spark.table("myhive.courier\_feedback\_clean").show()

spark.table("myhive.courier\_report").show()

spark.table("myhive.courier\_report\_clean").show()

spark.stop()

#database\_input\_and\_create.sql create database if not exists myhive;

use myhive;

create table myhive.courier\_device( no int comment '序号', city\_id string comment '城市ID', day int comment '天数', courier\_id string comment '快递员ID哈希值', os\_version string comment '系统版本', phone\_mode string comment '手机型号' )row format delimited fields terminated by ',' tblproperties ('skip.header.line.count'='1');

create table myhive.merchant\_device( no int comment '序号', city\_id string comment '城市ID', day int comment '天数', shop\_id string comment '商家ID哈希值', beacon\_id string comment '信标ID哈希值', manufacturer string comment '制造商', brand string comment '品牌', model string comment '型号' )row format delimited fields terminated by ',' tblproperties ('skip.header.line.count'='1');

create table myhive.courier\_report( no int comment '序号', city\_id string comment '城市ID', day int comment '天数', courier\_id string comment '快递员ID哈希值', shop\_id string comment '商家ID哈希值', acceptance\_hour int comment '快递员接受订单时间（小时）', acceptance\_minute int comment '快递员接受订单时间（分钟）', acceptance\_second int comment '快递员接受订单时间（秒）', arrier\_hour int comment '快递员到达商户时间（小时）', arrier\_minute int comment '快递员到达商户时间（分钟）', arrier\_second int comment '快递员到达商户时间（秒）', pickup\_hour int comment '快递员取货时间（小时）', pickup\_minute int comment '快递员取货时间（分钟）', pickup\_second int comment '快递员取货时间（秒）', delivery\_hour int comment '快递完成时间（小时）', delivery\_minute int comment '快递完成时间（分钟）', delivery\_second int comment '快递完成时间（秒）' )row format delimited fields terminated by ',' tblproperties ('skip.header.line.count'='1');

create table myhive.courier\_feedback( city\_id string comment '城市ID', day int comment '天数', courier\_id string comment '快递员ID哈希值', shop\_id string comment '商家ID哈希值', acceptance\_hour int comment '快递员接受订单时间（小时）', acceptance\_minute int comment '快递员接受订单时间（分钟）', acceptance\_second int comment '快递员接受订单时间（秒）', feedback\_hour int comment '快递员响应通知时间（小时）', feedback\_minute int comment '快递员响应通知时间（分钟）', feedback\_second int comment '快递员响应通知时间（秒）', feedback int comment '快递员对通知的回应（0是稍后尝试，1是确认）', distance\_to\_merchant double comment '响应通知时快递员和商店的距离', gps\_measure\_offset int comment '反馈时间和gps时间的时间差' )row format delimited fields terminated by ',' tblproperties ('skip.header.line.count'='1');

load data inpath '/tmp/file/Courier\_device\_data.csv' into table myhive.courier\_device;

load data inpath '/tmp/file/Merchant\_device\_data.csv' into table myhive.merchant\_device;

load data inpath '/tmp/file/Courier\_feedback\_data.csv/Courier\_feedback\_data\_city\_1.csv' into table myhive.courier\_feedback; load data inpath '/tmp/file/Courier\_feedback\_data.csv/Courier\_feedback\_data\_city\_2.csv' into table myhive.courier\_feedback; load data inpath '/tmp/file/Courier\_feedback\_data.csv/Courier\_feedback\_data\_city\_3.csv' into table myhive.courier\_feedback; load data inpath '/tmp/file/Courier\_feedback\_data.csv/Courier\_feedback\_data\_city\_4.csv' into table myhive.courier\_feedback; load data inpath '/tmp/file/Courier\_feedback\_data.csv/Courier\_feedback\_data\_city\_5.csv' into table myhive.courier\_feedback; load data inpath '/tmp/file/Courier\_feedback\_data.csv/Courier\_feedback\_data\_city\_6.csv' into table myhive.courier\_feedback; load data inpath '/tmp/file/Courier\_feedback\_data.csv/Courier\_feedback\_data\_city\_7.csv' into table myhive.courier\_feedback; load data inpath '/tmp/file/Courier\_feedback\_data.csv/Courier\_feedback\_data\_city\_8.csv' into table myhive.courier\_feedback;

load data inpath '/tmp/file/Courier\_report\_data.csv/Courier\_report\_data\_city\_1.csv' into table myhive.courier\_report; load data inpath '/tmp/file/Courier\_report\_data.csv/Courier\_report\_data\_city\_2.csv' into table myhive.courier\_report; load data inpath '/tmp/file/Courier\_report\_data.csv/Courier\_report\_data\_city\_3.csv' into table myhive.courier\_report; load data inpath '/tmp/file/Courier\_report\_data.csv/Courier\_report\_data\_city\_4.csv' into table myhive.courier\_report; load data inpath '/tmp/file/Courier\_report\_data.csv/Courier\_report\_data\_city\_5.csv' into table myhive.courier\_report; load data inpath '/tmp/file/Courier\_report\_data.csv/Courier\_report\_data\_city\_6.csv' into table myhive.courier\_report; load data inpath '/tmp/file/Courier\_report\_data.csv/Courier\_report\_data\_city\_7.csv' into table myhive.courier\_report; load data inpath '/tmp/file/Courier\_report\_data.csv/Courier\_report\_data\_city\_8.csv' into table myhive.courier\_report; load data inpath '/tmp/file/Courier\_report\_data.csv/Courier\_report\_data\_city\_9.csv' into table myhive.courier\_report;

-- create table myhive.courier\_device\_clean( -- no int comment '序号', -- city\_id string comment '城市ID', -- day int comment '天数', -- courier\_id string comment '快递员ID哈希值', -- os\_version string comment '系统版本', -- phone\_mode string comment '手机型号' -- )row format delimited fields terminated by ','; -- -- create table myhive.merchant\_device\_clean( -- no int comment '序号', -- city\_id string comment '城市ID', -- day int comment '天数', -- shop\_id string comment '商家ID哈希值', -- beacon\_id string comment '信标ID哈希值', -- manufacturer string comment '制造商', -- brand string comment '品牌', -- model string comment '型号' -- )row format delimited fields terminated by ','; -- -- create table myhive.courier\_report\_clean( -- no int comment '序号', -- city\_id string comment '城市ID', -- day int comment '天数', -- courier\_id string comment '快递员ID哈希值', -- shop\_id string comment '商家ID哈希值', -- acceptance\_hour int comment '快递员接受订单时间（小时）', -- acceptance\_minute int comment '快递员接受订单时间（分钟）', -- acceptance\_second int comment '快递员接受订单时间（秒）', -- arrier\_hour int comment '快递员到达商户时间（小时）', -- arrier\_minute int comment '快递员到达商户时间（分钟）', -- arrier\_second int comment '快递员到达商户时间（秒）', -- pickup\_hour int comment '快递员取货时间（小时）', -- pickup\_minute int comment '快递员取货时间（分钟）', -- pickup\_second int comment '快递员取货时间（秒）', -- delivery\_hour int comment '快递完成时间（小时）', -- delivery\_minute int comment '快递完成时间（分钟）', -- delivery\_second int comment '快递完成时间（秒）' -- )row format delimited fields terminated by ','; -- -- create table myhive.courier\_feedback\_clean( -- city\_id string comment '城市ID', -- day int comment '天数', -- courier\_id string comment '快递员ID哈希值', -- shop\_id string comment '商家ID哈希值', -- acceptance\_hour int comment '快递员接受订单时间（小时）', -- acceptance\_minute int comment '快递员接受订单时间（分钟）', -- acceptance\_second int comment '快递员接受订单时间（秒）', -- feedback\_hour int comment '快递员响应通知时间（小时）', -- feedback\_minute int comment '快递员响应通知时间（分钟）', -- feedback\_second int comment '快递员响应通知时间（秒）', -- feedback int comment '快递员对通知的回应（0是稍后尝试，1是确认）', -- distance\_to\_merchant double comment '响应通知时快递员和商店的距离', -- gps\_measure\_offset int comment '反馈时间和gps时间的时间差' -- )row format delimited fields terminated by ',';

#courier\_analysis.py from pyspark.sql.types import Row, DoubleType, StructType, StructField from pyspark.sql import SparkSession from pyspark.sql.functions import col, desc, unix\_timestamp, lpad, concat\_ws, when, expr

if **name** == '**main**': spark = SparkSession.builder.   
appName("courier\_analysis").   
master("yarn").   
config("spark.sql.shuffle.partitions", 2).   
config("spark.sql.warehouse.dir", "hdfs://node1:8020/user/hive/warehouse").   
config("hive.metastore.uris", "thrift://node1:9083").   
enableHiveSupport().   
getOrCreate() sc = spark.sparkContext

# 将feedback改成中文

feedback\_mapping = {

0: "稍后尝试",

1: "确认"

}

feedback\_mapping\_df = spark.createDataFrame([Row(feedback=k, feedback\_tran=v) for k, v in feedback\_mapping.items()])

# 读取表数据

courier\_feedback\_df = spark.table("myhive.courier\_feedback\_clean")

courier\_report\_df = spark.table("myhive.courier\_report\_clean")

# 计算响应时间（以秒为单位）

response\_time\_seconds = (col("feedback\_hour") \* 3600 + col("feedback\_minute") \* 60 + col("feedback\_second")) - \

(col("acceptance\_hour") \* 3600 + col("acceptance\_minute") \* 60 + col("acceptance\_second"))

# 添加响应时间列到 DataFrame

courier\_feedback\_with\_response\_time\_df = courier\_feedback\_df.withColumn("response\_time\_seconds",

response\_time\_seconds)

courier\_feedback\_with\_response\_time\_df = courier\_feedback\_with\_response\_time\_df.join(feedback\_mapping\_df,

on=["feedback"],

how="inner")

# 计算平均响应时间

avg\_response\_time = courier\_feedback\_with\_response\_time\_df.agg({"response\_time\_seconds": "avg"}).collect()[0][0]

print("平均响应时间: {:.2f} 秒".format(avg\_response\_time))

# 手动定义模式

schema = StructType([

StructField("avg\_response\_second", DoubleType()),

StructField("avg\_response\_minute", DoubleType())

])

# 将平均响应时间写入hive表

spark.createDataFrame([(avg\_response\_time, avg\_response\_time / 60)], schema) \

.write.mode("overwrite").option("delimiter", ",").format("orc").saveAsTable("myhive\_detail.avg\_response\_time")

# 分析快递员对通知的回应情况

response\_counts = courier\_feedback\_with\_response\_time\_df.groupBy("courier\_id", "feedback\_tran").count() \

.withColumnRenamed("count", "feedback\_count")

response\_counts.where("feedback\_tran = '稍后尝试'").orderBy(desc("feedback\_count")).show()

response\_counts.where("feedback\_tran = '确认'").orderBy(desc("feedback\_count")).show()

# 将分析快递员对通知的回应情况写入hive表

response\_counts.write.mode("overwrite").option("delimiter", ",").format("orc").saveAsTable(

"myhive\_detail.courier\_response\_counts")

# 分析快递员处理的订单数量

order\_counts = courier\_feedback\_with\_response\_time\_df.groupBy("courier\_id").count() \

.withColumnRenamed("count", "order\_count")

order\_counts.orderBy(desc("order\_count")).show()

# 计算每个外卖员的工作时长（以秒为单位）

courier\_report\_df = courier\_report\_df.withColumn("acceptance\_time",

concat\_ws(':', lpad(col("acceptance\_hour"), 2, '0'),

lpad(col("acceptance\_minute"), 2, '0'),

lpad(col("acceptance\_second"), 2, '0')))

courier\_report\_df = courier\_report\_df.withColumn("delivery\_time",

concat\_ws(':', lpad(col("delivery\_hour"), 2, '0'),

lpad(col("delivery\_minute"), 2, '0'),

lpad(col("delivery\_second"), 2, '0')))

# 将时间字符串转换为时间戳，并计算时间差（秒）

work\_duration\_seconds = (

unix\_timestamp(col("delivery\_time"), "HH:mm:ss") -

unix\_timestamp(col("acceptance\_time"), "HH:mm:ss")).cast("bigint")

# 考虑跨越天数的情况，处理负数时长

work\_duration\_seconds = when(work\_duration\_seconds < 0, 24 \* 3600 + work\_duration\_seconds).otherwise(

work\_duration\_seconds)

# 添加工作时长列到 DataFrame

courier\_report\_with\_work\_duration\_df = courier\_report\_df.withColumn("work\_duration\_seconds", work\_duration\_seconds)

# 计算每个外卖员的平均工作时长（以秒为单位）

avg\_work\_duration\_seconds = courier\_report\_with\_work\_duration\_df.groupBy("courier\_id") \

.agg({"work\_duration\_seconds": "avg"}) \

.withColumnRenamed("avg(work\_duration\_seconds)", "avg\_work\_duration\_seconds")

avg\_work\_duration\_seconds.show()

# 将快递员数据整合起来

courier\_final = order\_counts.join(avg\_work\_duration\_seconds, on=["courier\_id"], how="inner")

courier\_final.show()

# 计算avg\_work\_duration\_seconds和order\_count列的99%和1%分位数

percentiles = courier\_final.approxQuantile(["avg\_work\_duration\_seconds", "order\_count"], [0.01, 0.99], 0.001)

# 提取分位数

p01\_work\_duration\_seconds, p99\_work\_duration\_seconds = percentiles[0]

p10\_order\_count, p90\_order\_count = percentiles[1]

# 使用when函数将每列的值映射到0到100的范围内

courier\_final = courier\_final.withColumn("work\_duration\_score",

expr("CASE WHEN avg\_work\_duration\_seconds <= {} THEN 100 "

"WHEN avg\_work\_duration\_seconds >= {} THEN 0 "

"ELSE 100 \* ({} - avg\_work\_duration\_seconds) / ({} - {}) END"

.format(p01\_work\_duration\_seconds, p99\_work\_duration\_seconds,

p99\_work\_duration\_seconds, p99\_work\_duration\_seconds,

p01\_work\_duration\_seconds)))

courier\_final = courier\_final.withColumn("order\_count\_score",

expr("CASE WHEN order\_count <= {} THEN 0 "

"WHEN order\_count >= {} THEN 100 "

"ELSE 100 \* (order\_count - {}) / ({} - {}) END"

.format(p10\_order\_count, p90\_order\_count,

p10\_order\_count, p90\_order\_count,

p10\_order\_count)))

# 计算得分的平均值

courier\_final = courier\_final.withColumn("final\_score",

(col("work\_duration\_score") \* 0.4) + (col("order\_count\_score") \* 0.6))

courier\_final = courier\_final.drop(\*('work\_duration\_score', 'order\_count\_score'))

courier\_final.orderBy(desc("final\_score")).show()

# 将快递员数据整合写入hive表

courier\_final.write.mode("overwrite").option("delimiter", ",").format("orc").saveAsTable(

"myhive\_detail.courier\_final")

# 停止SparkSession

spark.stop()

#delivery\_time.py from pyspark.sql import SparkSession from pyspark.sql.functions import col, concat\_ws, unix\_timestamp, lpad, avg, desc, rank, count from pyspark.sql.types import Row from pyspark.sql.window import Window

if **name** == '**main**': spark = SparkSession.builder.   
appName("delivery\_time").   
master("yarn").   
config("spark.sql.shuffle.partitions", 2).   
config("spark.sql.warehouse.dir", "hdfs://node1:8020/user/hive/warehouse").   
config("hive.metastore.uris", "thrift://node1:9083").   
enableHiveSupport().   
getOrCreate() sc = spark.sparkContext

# 创建city\_mapping字典

city\_mapping = {

1: '上海',

2: '杭州',

3: '深圳',

4: '北京',

5: '潍坊',

6: '广州',

7: '遂宁',

8: '西宁',

9: '定安',

10: '廊坊'

}

# 将字典转换为DataFrame

city\_mapping\_df = spark.createDataFrame([Row(city\_id=k, city\_name=v) for k, v in city\_mapping.items()])

# 读取Hive表

courier\_report\_df = spark.table("myhive.courier\_report\_clean")

# 计算接受订单和完成配送的时间戳

courier\_report\_df = courier\_report\_df.withColumn("acceptance\_time",

concat\_ws(':', lpad(col("acceptance\_hour"), 2, '0'),

lpad(col("acceptance\_minute"), 2, '0'),

lpad(col("acceptance\_second"), 2, '0')))

courier\_report\_df = courier\_report\_df.withColumn("delivery\_time",

concat\_ws(':', lpad(col("delivery\_hour"), 2, '0'),

lpad(col("delivery\_minute"), 2, '0'),

lpad(col("delivery\_second"), 2, '0')))

# 将时间字符串转换为时间戳，并计算时间差（秒）

courier\_report\_df = courier\_report\_df.withColumn("total\_delivery\_time\_seconds",

(unix\_timestamp(col("delivery\_time"), "HH:mm:ss") - unix\_timestamp(

col("acceptance\_time"), "HH:mm:ss")).cast("bigint"))

# 显示结果

courier\_report\_df.show()

# 分析配送流程效率，例如计算平均配送时间

average\_delivery\_time = courier\_report\_df.agg({"total\_delivery\_time\_seconds": "avg"}).collect()[0][0]

print(f"平均配送时间: {average\_delivery\_time} 秒（{average\_delivery\_time / 60} 分钟）")

# 将平均配送时间写入Hive表

spark.createDataFrame([(average\_delivery\_time, average\_delivery\_time / 60)],

["average\_delivery\_second", "average\_delivery\_minute"]) \

.write.mode("overwrite").option("delimiter", ",").format("orc").saveAsTable(

"myhive\_detail.average\_delivery\_time")

# 聚合分析，按小时计算平均配送时间

average\_delivery\_time\_per\_hour = courier\_report\_df.groupBy("city\_id", "acceptance\_hour") \

.agg(avg("total\_delivery\_time\_seconds").alias("avg\_delivery\_time\_seconds"),

count("total\_delivery\_time\_seconds").alias("count")).where("count > 1000")

# 定义窗口规范，按城市进行分区，并按订单数量降序排序

window\_spec = Window.partitionBy("city\_id").orderBy(desc("avg\_delivery\_time\_seconds"))

# 添加排名列

ranked\_df = average\_delivery\_time\_per\_hour.withColumn("rank", rank().over(window\_spec))

# 过滤出每个城市排名在前三的时间段

top\_busy\_hours\_per\_city\_df = ranked\_df.where(col("rank") <= 3).orderBy(desc("avg\_delivery\_time\_seconds")) \

.join(city\_mapping\_df, on=["city\_id"], how="inner") \

.select("city\_name", "acceptance\_hour", "avg\_delivery\_time\_seconds", "count")

top\_busy\_hours\_per\_city\_df.show()

# 存到hive表

top\_busy\_hours\_per\_city\_df.write.mode("overwrite").option("delimiter", ",").format("orc").saveAsTable(

"myhive\_detail.top\_busy\_hours\_per\_city")

# 停止SparkSession

spark.stop()

#phone\_model.py from pyspark.sql import SparkSession from pyspark.sql.types import IntegerType, Row

if **name** == '**main**': spark = SparkSession.builder.   
appName("phone\_model").   
master("yarn").   
config("spark.sql.shuffle.partitions", 2).   
config("spark.sql.warehouse.dir", "hdfs://node1:8020/user/hive/warehouse").   
config("hive.metastore.uris", "thrift://node1:9083").   
enableHiveSupport().   
getOrCreate() sc = spark.sparkContext

df\_courier = spark.table("myhive.courier\_device\_clean")

df\_courier = df\_courier.withColumn("city\_id", df\_courier["city\_id"].cast(IntegerType()))

df\_courier.createOrReplaceTempView("courier\_view")

df\_merchant = spark.table("myhive.merchant\_device\_clean")

df\_merchant = df\_merchant.withColumn("city\_id", df\_merchant["city\_id"].cast(IntegerType()))

df\_merchant.createOrReplaceTempView("shop\_view")

# 创建city\_mapping字典

city\_mapping = {

1: '上海',

2: '杭州',

3: '深圳',

4: '北京',

5: '潍坊',

6: '广州',

7: '遂宁',

8: '西宁',

9: '定安',

10: '廊坊'

}

# 将字典转换为DataFrame

city\_mapping\_df = spark.createDataFrame([Row(city\_id=k, city\_name=v) for k, v in city\_mapping.items()])

# 统计快递员操作系统版本和手机型号的使用频率

print("courier\_phone\_mode\_counts")

courier\_phone\_mode\_counts = spark.sql("""

SELECT phone\_mode, COUNT(\*) as count

FROM courier\_view

GROUP BY phone\_mode

ORDER BY count DESC

LIMIT 20

""")

courier\_phone\_mode\_counts.show()

# 将数据存入hive中

courier\_phone\_mode\_counts.write.mode("overwrite").option("delimiter", ",").format("orc") \

.saveAsTable("myhive\_detail.courier\_phone\_mode\_counts")

print("courier\_os\_version\_counts")

courier\_os\_version\_counts = spark.sql("""

SELECT phone\_mode, os\_version, COUNT(\*) as count

FROM courier\_view

GROUP BY phone\_mode, os\_version

ORDER BY count DESC

LIMIT 20

""")

courier\_os\_version\_counts.show()

# 将数据存入hive中

courier\_os\_version\_counts.write.mode("overwrite").option("delimiter", ",").format("orc") \

.saveAsTable("myhive\_detail.courier\_os\_version\_counts")

print("courier\_city\_phone\_mode\_favourite")

courier\_city\_phone\_mode\_favourite = spark.sql("""

SELECT city\_id, phone\_mode, count

FROM (

SELECT city\_id, phone\_mode, COUNT(\*) as count,

ROW\_NUMBER() OVER (PARTITION BY city\_id ORDER BY COUNT(\*) DESC) as rn

FROM courier\_view

GROUP BY city\_id, phone\_mode

) t

WHERE t.rn = 1

order by city\_id ASC

LIMIT 20

""").join(city\_mapping\_df, on=["city\_id"], how="inner").select("city\_name", "phone\_mode", "count")

courier\_city\_phone\_mode\_favourite.show()

# 将数据存入hive中

courier\_city\_phone\_mode\_favourite.write.mode("overwrite").option("delimiter", ",").format("orc") \

.saveAsTable("myhive\_detail.courier\_city\_phone\_mode\_favourite")

# 统计商家制造商和品牌的使用频率

print("shop\_brand\_counts")

shop\_brand\_counts = spark.sql("""

SELECT brand, COUNT(\*) as count

FROM shop\_view

GROUP BY brand

ORDER BY count DESC

LIMIT 20

""")

shop\_brand\_counts.show()

# 将数据存入hive中

shop\_brand\_counts.write.mode("overwrite").option("delimiter", ",").format("orc") \

.saveAsTable("myhive\_detail.shop\_brand\_counts")

print("shop\_model\_counts")

shop\_model\_counts = spark.sql("""

SELECT model, COUNT(\*) as count

FROM shop\_view

GROUP BY model

ORDER BY count DESC

LIMIT 20

""")

shop\_model\_counts.show()

# 将数据存入hive中

shop\_model\_counts.write.mode("overwrite").option("delimiter", ",").format("orc") \

.saveAsTable("myhive\_detail.shop\_model\_counts")

print("shop\_city\_model\_favourite")

shop\_city\_model\_favourite = spark.sql("""

SELECT city\_id, model, count

FROM (

SELECT city\_id, model, COUNT(\*) as count,

ROW\_NUMBER() OVER (PARTITION BY city\_id ORDER BY COUNT(\*) DESC) as rn

FROM shop\_view

GROUP BY city\_id, model

)

WHERE t.rn = 1

order by city\_id ASC

LIMIT 20

""").join(city\_mapping\_df, on=["city\_id"], how="inner").select("city\_name", "model", "count")

shop\_city\_model\_favourite.show()

# 将数据存入hive中

shop\_city\_model\_favourite.write.mode("overwrite").option("delimiter", ",").format("orc") \

.saveAsTable("myhive\_detail.shop\_city\_model\_favourite")

# 对每个城市的外卖员和商家数量统计

courier\_counts = spark.sql("SELECT city\_id, COUNT(\*) AS courier\_count FROM courier\_view GROUP BY city\_id")

merchant\_counts = spark.sql("SELECT city\_id, COUNT(\*) AS merchant\_count FROM shop\_view GROUP BY city\_id")

city\_counts = courier\_counts.join(merchant\_counts, "city\_id", "inner") \

.join(city\_mapping\_df, on=["city\_id"], how="inner").select("city\_name", "courier\_count", "merchant\_count")

city\_counts.show()

city\_counts.write.mode("overwrite").option("delimiter", ",").format("orc").saveAsTable(

"myhive\_detail.city\_courier\_merchant\_counts")

spark.stop()

#shop\_analysis.py from pyspark.sql import SparkSession from pyspark.sql.window import Window from pyspark.sql.functions import desc, rank, col

if **name** == '**main**': spark = SparkSession.builder.   
appName("shop\_analysis").   
master("yarn").   
config("spark.sql.shuffle.partitions", 2).   
config("spark.sql.warehouse.dir", "hdfs://node1:8020/user/hive/warehouse").   
config("hive.metastore.uris", "thrift://node1:9083").   
enableHiveSupport().   
getOrCreate() sc = spark.sparkContext

# 读取表数据

courier\_feedback\_df = spark.table("myhive.courier\_feedback\_clean")

courier\_report\_df = spark.table("myhive.courier\_report\_clean")

# 分析商家处理的订单数量

shop\_order\_counts = courier\_feedback\_df.groupBy("shop\_id").count() \

.withColumnRenamed("count", "order\_count")

shop\_order\_counts.orderBy(desc("order\_count")).show()

# 写入到hive表

shop\_order\_counts.write.mode("overwrite").option("delimiter", ",").format("orc").saveAsTable(

"myhive\_detail.shop\_order\_counts")

# 分析每天最高销量的三个商家

shop\_top\_by\_day\_df = courier\_feedback\_df.groupBy(["day", "shop\_id"]).count() \

.withColumnRenamed("count", "shop\_day\_count")

# 定义窗口规范，按城市进行分区，并按订单数量降序排序

window\_spec = Window.partitionBy("day").orderBy(desc("shop\_day\_count"))

# 添加排名列

ranked\_df = shop\_top\_by\_day\_df.withColumn("rank", rank().over(window\_spec))

# 过滤出每个城市排名在前三的时间段

shop\_top\_by\_day = ranked\_df.where(col("rank") <= 3).orderBy(desc("shop\_day\_count"))

shop\_top\_by\_day = shop\_top\_by\_day.orderBy(col("day").asc(), col("rank").asc()).select("day", "shop\_id",

"shop\_day\_count")

shop\_top\_by\_day.show()

# 写入到hive表

shop\_top\_by\_day.write.mode("overwrite").option("delimiter", ",").format("orc").saveAsTable(

"myhive\_detail.shop\_top\_by\_day")

# 结束spark

spark.stop()

#courier\_delivery\_time\_prediction.py import time from pyspark.sql import SparkSession from pyspark.ml.feature import VectorAssembler, OneHotEncoder, MinMaxScaler from pyspark.ml.regression import GBTRegressor, RandomForestRegressor, LinearRegression from pyspark.ml.evaluation import RegressionEvaluator from pyspark.sql.functions import when, lit, col from pyspark.sql import DataFrame

**评价模型**

def evaluate\_predictions(predictions, name):

**计算MAE、MSE、RMSE和R-Squared等**

evaluator = RegressionEvaluator(labelCol="delivery\_time", predictionCol=name) mae = evaluator.evaluate(predictions, {evaluator.metricName: "mae"}) mse = evaluator.evaluate(predictions, {evaluator.metricName: "mse"}) rmse = evaluator.evaluate(predictions, {evaluator.metricName: "rmse"}) r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})

# 输出指标结果

print(f"MAE:{mae} MSE:{mse} RMSE:{rmse} R-Squared:{r2}")

return mae, mse, rmse, r2

**将时间转换成时间戳**

def convert\_to\_timestamp(df: DataFrame, time\_cols: list) -> DataFrame: for col\_name in time\_cols: df = df.withColumn(f"{col\_name}\_time", col(f"{col\_name}\_hour") \* 60 + col(f"{col\_name}\_minute")) df = df.withColumn(f"{col\_name}\_time", col(f"{col\_name}\_time").cast("int")) if col\_name != "acceptance": df = df.withColumn(f"{col\_name}\_time", when((col("acceptance\_time") > 1080) & (col(f"{col\_name}\_time") < 120), col(f"{col\_name}\_time") + 1440).otherwise( col(f"{col\_name}\_time")) ) return df

if **name** == '**main**': spark = SparkSession.builder.   
appName("courier\_delivery\_time\_prediction").   
master("yarn").   
config("spark.sql.shuffle.partitions", 2).   
config("spark.sql.warehouse.dir", "hdfs://node1:8020/user/hive/warehouse").   
config("hive.metastore.uris", "thrift://node1:9083").   
enableHiveSupport().   
getOrCreate() sc = spark.sparkContext

# 读取快递报告数据

courier\_report\_df = spark.table("myhive.courier\_report\_clean")

courier\_feedback\_df = spark.table("myhive.courier\_feedback\_clean")

# 删除非必要数据

courier\_report\_df = courier\_report\_df.drop("no")

# 使用预测的数据

data\_df = courier\_report\_df.join(courier\_feedback\_df,

on=["city\_id", "day", "courier\_id", "shop\_id", "acceptance\_hour",

"acceptance\_minute",

"acceptance\_second"],

how="left")

data\_df = data\_df.withColumn("has\_system\_notification",

when((data\_df["feedback\_hour"].isNull())

, lit(0)).otherwise(lit(1)))

data\_df = data\_df.fillna({"feedback": 2, "distance\_to\_merchant": 0, "gps\_measure\_offset": 0, "feedback\_hour": 24,

"feedback\_minute": 0, "feedback\_second": 0})

data\_df = data\_df.withColumn("city\_id", col("city\_id").cast("int"))

# 调用函数进行转换

time\_columns = ["acceptance", "arrier", "pickup", "feedback", "delivery"]

data\_df = convert\_to\_timestamp(data\_df, time\_columns)

data\_df = data\_df.drop(\*(

"acceptance\_hour", "acceptance\_minute", "acceptance\_second", "arrier\_hour", "arrier\_minute", "arrier\_second",

"pickup\_hour", "pickup\_minute", "pickup\_second", "feedback\_hour", "feedback\_minute", "feedback\_second",

"delivery\_hour", "delivery\_minute", "delivery\_second"))

# 进一步处理特征向量

# 创建OneHotEncoder将索引编码成独热向量

encoder = OneHotEncoder(inputCol="city\_id", outputCol="city\_id\_encoded")

data\_df = encoder.fit(data\_df).transform(data\_df)

data\_df.show()

# 特征选择

feature\_columns = ['city\_id\_encoded', "day", 'acceptance\_time', 'arrier\_time', 'pickup\_time', "feedback\_time",

"feedback",

"distance\_to\_merchant", "gps\_measure\_offset", "has\_system\_notification"]

# 合并特征列为一个特征向量

assembler = VectorAssembler(inputCols=feature\_columns, outputCol="raw\_features")

data = assembler.transform(data\_df)

data.show()

# 归一化特征向量

scaler = MinMaxScaler(inputCol="raw\_features", outputCol="features")

scaler\_model = scaler.fit(data)

data = scaler\_model.transform(data)

# 划分数据集为训练集和测试集

train\_data, test\_data = data.randomSplit([0.8, 0.2], seed=42)

# 初始化线性回归模型

lr\_model = LinearRegression(featuresCol="features", labelCol="delivery\_time", predictionCol="prediction\_lr")

# 获取训练开始时间

start\_time = time.time()

# 训练模型

lr\_trained\_model = lr\_model.fit(train\_data)

# 获取训练结束时间

end\_time = time.time()

# 计算训练时间

training\_time = end\_time - start\_time

print("线性回归训练时间:", training\_time, "seconds")

# 在测试集上进行预测

lr\_predictions = lr\_trained\_model.transform(test\_data)

lr\_predictions\_train = lr\_trained\_model.transform(train\_data)

# 评估模型性能

print("线性回归 test\_data 评估")

evaluate\_predictions(lr\_predictions, "prediction\_lr")

print("线性回归 train\_data 评估")

evaluate\_predictions(lr\_predictions\_train, "prediction\_lr")

# 初始化随机森林回归模型

rf\_model = RandomForestRegressor(featuresCol="features", labelCol="delivery\_time",

predictionCol="prediction\_rf")

# 获取训练开始时间

start\_time = time.time()

# 训练模型

trained\_model = rf\_model.fit(train\_data)

# 获取训练结束时间

end\_time = time.time()

# 计算训练时间

training\_time = end\_time - start\_time

print("随机森林训练时间:", training\_time, "seconds")

# 在测试集上进行预测

rf\_predictions = trained\_model.transform(test\_data)

rf\_predictions\_train = trained\_model.transform(train\_data)

# 评估模型性能

print("随机森林test\_data评估")

evaluate\_predictions(rf\_predictions, "prediction\_rf")

print("随机森林train\_data评估")

evaluate\_predictions(rf\_predictions\_train, "prediction\_rf")

# 初始化 GBTRegressor 模型

gbt = GBTRegressor(featuresCol="features", labelCol="delivery\_time", predictionCol="prediction\_gbt")

# 获取训练开始时间

start\_time = time.time()

# 训练模型

gbt\_model = gbt.fit(train\_data)

# 获取训练结束时间

end\_time = time.time()

# 计算训练时间

training\_time = end\_time - start\_time

print("GBT训练时间:", training\_time, "seconds")

# 在测试集上进行预测

gbt\_predictions = gbt\_model.transform(test\_data)

gbt\_predictions\_train = gbt\_model.transform(train\_data)

# 评估模型

# 评估模型性能

print("GBT test\_data评估")

evaluate\_predictions(gbt\_predictions, "prediction\_gbt")

print("GBT train\_data评估")

evaluate\_predictions(gbt\_predictions\_train, "prediction\_gbt")

# 使用均值融合模型

fusion\_predictions = rf\_predictions.join(gbt\_predictions, on=["city\_id", "day", "courier\_id", "shop\_id", "feedback",

"distance\_to\_merchant", "gps\_measure\_offset",

"has\_system\_notification", "acceptance\_time",

"arrier\_time", "pickup\_time", "feedback\_time",

"delivery\_time", "city\_id\_encoded", "features"],

how="inner")

fusion\_predictions = fusion\_predictions.join(lr\_predictions,

on=["city\_id", "day", "courier\_id", "shop\_id", "feedback",

"distance\_to\_merchant", "gps\_measure\_offset",

"has\_system\_notification", "acceptance\_time",

"arrier\_time", "pickup\_time", "feedback\_time",

"delivery\_time", "city\_id\_encoded", "features"],

how="inner") \

.withColumn("fusion\_prediction", (col("prediction\_rf") + col("prediction\_gbt") + col("prediction\_lr")) / 3.0)

print("融合模型test\_data评估")

evaluate\_predictions(fusion\_predictions, "fusion\_prediction")

fusion\_predictions\_train = rf\_predictions\_train.join(gbt\_predictions\_train,

on=["city\_id", "day", "courier\_id", "shop\_id", "feedback",

"distance\_to\_merchant", "gps\_measure\_offset",

"has\_system\_notification", "acceptance\_time",

"arrier\_time", "pickup\_time", "feedback\_time",

"delivery\_time", "city\_id\_encoded", "features"],

how="inner")

fusion\_predictions\_train = fusion\_predictions\_train.join(lr\_predictions\_train,

on=["city\_id", "day", "courier\_id", "shop\_id", "feedback",

"distance\_to\_merchant", "gps\_measure\_offset",

"has\_system\_notification", "acceptance\_time",

"arrier\_time", "pickup\_time", "feedback\_time",

"delivery\_time", "city\_id\_encoded", "features"],

how="inner") \

.withColumn("fusion\_prediction", (col("prediction\_rf") + col("prediction\_gbt") + col("prediction\_lr")) / 3.0)

print("融合模型train\_data评估")

evaluate\_predictions(fusion\_predictions\_train, "fusion\_prediction")

# 结束 SparkSession

spark.stop()