FIT5202 Assignment 2A : Building Models for Realtime Food Delivery Prediction

Table of Contents

Please add code/markdown cells as needed.

Part 1: Data Loading, Transformation and Exploration

1.1 Data Loading

In this section, you must load the given datasets into PySpark DataFrames and use DataFrame functions to process the data. Spark SQL usage is discouraged, and you can only use pandas to format results. For plotting, various visualisation packages can be used, but please ensure that you have included instructions to install the additional packages and that the installation will be successful in the provided docker container (in case your marker needs to clear the notebook and rerun it).

1.1.1 Data Loading

1.1.1 Write the code to create a SparkSession. Please use a SparkConf object to configure the Spark app with a proper application name, to ensure the maximum partition size does not exceed 16MB, and to run locally with 4 CPU cores on your machine .

In [17]:
```
from pyspark.sql import SparkSession
from pyspark import SparkConf

# Configure the Spark application
conf = SparkConf()
conf.setAppName("Realtime_Food_Delivery_Prediction")  # Application name
conf.set("spark.sql.files.maxPartitionBytes", "16777216")  # Set max partition s
conf.set("spark.executor.memory", "2g")  # Set executor memory (adjustable based
conf.set("spark.driver.memory", "2g")  # Set driver memory (adjustable based on
conf.setMaster("local[4]")  # Run locally with 4 CPU cores

# Create a SparkSession
spark = SparkSession.builder.config(conf=conf).getOrCreate()

# Print Spark configuration to verify settings
print("Spark Configuration:")
for item in spark.sparkContext.getConf().getAll():
    print(f"{item[0]}: {item[1]}")
```

```python
# Verify SparkSession creation
print("\nSparkSession successfully created!")
```

```
Spark Configuration:
spark.executor.memory: 2g
spark.master: local[4]
spark.executor.id: driver
spark.app.submitTime: 1738280487589
spark.driver.host: 1f118851ebed
spark.app.startTime: 1738280487824
spark.sql.files.maxPartitionBytes: 16777216
spark.driver.extraJavaOptions: -Djava.net.preferIPv6Addresses=false -XX:+IgnoreUn
recognizedVMOptions --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.
base/java.lang.invoke=ALL-UNNAMED --add-opens=java.base/java.lang.reflect=ALL-UNN
AMED --add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.base/java.net=ALL
-UNNAMED --add-opens=java.base/java.nio=ALL-UNNAMED --add-opens=java.base/java.ut
il=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-opens
=java.base/java.util.concurrent.atomic=ALL-UNNAMED --add-opens=java.base/sun.nio.
ch=ALL-UNNAMED --add-opens=java.base/sun.nio.cs=ALL-UNNAMED --add-opens=java.bas
e/sun.security.action=ALL-UNNAMED --add-opens=java.base/sun.util.calendar=ALL-UNN
AMED --add-opens=java.security.jgss/sun.security.krb5=ALL-UNNAMED -Djdk.reflect.u
seDirectMethodHandle=false
spark.app.id: local-1738280489255
spark.app.name: Realtime_Food_Delivery_Prediction
spark.sql.warehouse.dir: file:/home/student/spark-warehouse
spark.rdd.compress: True
spark.driver.memory: 2g
spark.serializer.objectStreamReset: 100
spark.submit.pyFiles:
spark.submit.deployMode: client
spark.ui.showConsoleProgress: true
spark.driver.port: 42247
spark.executor.extraJavaOptions: -Djava.net.preferIPv6Addresses=false -XX:+Ignore
UnrecognizedVMOptions --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=jav
a.base/java.lang.invoke=ALL-UNNAMED --add-opens=java.base/java.lang.reflect=ALL-U
NNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.base/java.net=A
LL-UNNAMED --add-opens=java.base/java.nio=ALL-UNNAMED --add-opens=java.base/java.
util=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-ope
ns=java.base/java.util.concurrent.atomic=ALL-UNNAMED --add-opens=java.base/sun.ni
o.ch=ALL-UNNAMED --add-opens=java.base/sun.nio.cs=ALL-UNNAMED --add-opens=java.ba
se/sun.security.action=ALL-UNNAMED --add-opens=java.base/sun.util.calendar=ALL-UN
NAMED --add-opens=java.security.jgss/sun.security.krb5=ALL-UNNAMED -Djdk.reflect.
useDirectMethodHandle=false

SparkSession successfully created!
```

In this assessment all cells needs to run manualy one by one in order to successfully run all code.

1.1.2 Write code to define the schemas for the datasets, following the data types suggested in the metadata. Then, using predefined schemas, write code to load the CSV files into separate data frames. Print the schemas of all data frames.

In [98]:
```python
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,

# Define the schema for the delivery_address dataset
delivery_address_schema = StructType([
    StructField("gid", IntegerType(), True),
```

```
    StructField("street_name", StringType(), True),
    StructField("street_type", StringType(), True),
    StructField("suburb", StringType(), True),
    StructField("postcode", IntegerType(), True),
    StructField("state", StringType(), True),
    StructField("latitude", DoubleType(), True),
    StructField("longitude", DoubleType(), True),
    StructField("geom", StringType(), True),
    StructField("delivery_id", IntegerType(), True)
])

# Define the schema for the driver dataset
driver_schema = StructType([
    StructField("driver_id", IntegerType(), True),
    StructField("age", IntegerType(), True),
    StructField("rating", FloatType(), True),
    StructField("year_experience", IntegerType(), True),
    StructField("vehicle_condition", StringType(), True),
    StructField("type_of_vehicle", StringType(), True)
])

# Define the schema for the order dataset
order_schema = StructType([
    StructField("order_id", StringType(), True),
    StructField("delivery_person_id", IntegerType(), True),
    StructField("order_ts", LongType(), True),
    StructField("ready_ts", LongType(), True),
    StructField("weather_condition", StringType(), True),
    StructField("road_condition", StringType(), True),
    StructField("type_of_order", StringType(), True),
    StructField("order_total", IntegerType(), True),
    StructField("delivery_time", IntegerType(), True),
    StructField("travel_distance", FloatType(), True),
    StructField("restaurant_id", IntegerType(), True),
    StructField("delivery_id", IntegerType(), True)
])

# Define the schema for the restaurant dataset
restaurant_schema = StructType([
    StructField("row_id", IntegerType(), True),
    StructField("restaurant_code", StringType(), True),
    StructField("chain_id", StringType(), True),
    StructField("primary_cuisine", StringType(), True),
    StructField("latitude", DoubleType(), True),
    StructField("longitude", DoubleType(), True),
    StructField("geom", StringType(), True),
    StructField("restaurant_id", IntegerType(), True),
    StructField("suburb", StringType(), True),
    StructField("postcode", IntegerType(), True)
])
```

In [99]:

```
# File paths
delivery_address_path = 'delivery_address.csv'
driver_path = 'driver.csv'
order_path = 'order.csv'
restaurant_path = 'restaurants.csv'

# Load datasets
delivery_address_df = spark.read.csv(delivery_address_path, schema=delivery_addr
driver_df = spark.read.csv(driver_path, schema=driver_schema, header=True)
```

```
order_df = spark.read.csv(order_path, schema=order_schema, header=True)
restaurant_df = spark.read.csv(restaurant_path, schema=restaurant_schema, header

# Print schemas
print("Delivery Address Schema:")
delivery_address_df.printSchema()

print("\nDriver Schema:")
driver_df.printSchema()

print("\nOrder Schema:")
order_df.printSchema()

print("\nRestaurant Schema:")
restaurant_df.printSchema()
```

```
Delivery Address Schema:
root
 |-- gid: integer (nullable = true)
 |-- street_name: string (nullable = true)
 |-- street_type: string (nullable = true)
 |-- suburb: string (nullable = true)
 |-- postcode: integer (nullable = true)
 |-- state: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- geom: string (nullable = true)
 |-- delivery_id: integer (nullable = true)


Driver Schema:
root
 |-- driver_id: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- rating: float (nullable = true)
 |-- year_experience: integer (nullable = true)
 |-- vehicle_condition: string (nullable = true)
 |-- type_of_vehicle: string (nullable = true)


Order Schema:
root
 |-- order_id: string (nullable = true)
 |-- delivery_person_id: integer (nullable = true)
 |-- order_ts: long (nullable = true)
 |-- ready_ts: long (nullable = true)
 |-- weather_condition: string (nullable = true)
 |-- road_condition: string (nullable = true)
 |-- type_of_order: string (nullable = true)
 |-- order_total: integer (nullable = true)
 |-- delivery_time: integer (nullable = true)
 |-- travel_distance: float (nullable = true)
 |-- restaurant_id: integer (nullable = true)
 |-- delivery_id: integer (nullable = true)


Restaurant Schema:
root
 |-- row_id: integer (nullable = true)
 |-- restaurant_code: string (nullable = true)
 |-- chain_id: string (nullable = true)
 |-- primary_cuisine: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- geom: string (nullable = true)
 |-- restaurant_id: integer (nullable = true)
 |-- suburb: string (nullable = true)
 |-- postcode: integer (nullable = true)
```

1.2 Data Transformation to Create Features

Feature engineering involves transforming, combining or extracting information from the raw data to create more informative and relevant features that improve the performance of your ML models.

In our food delivery use case, the order_ts is not very useful when it is treated as a timestamp.
However, it provides more information if you perform transformation and extract valuable information from it, for example, extracting the day of the week (it may tell you how busy a restaurant is) or hours (peak hours may have bad traffic conditions).
(Note: Some tasks may overlap with A1, feel free to use/reuse your own code/UDF from A1.)

Perform the following tasks based on the loaded data frames and create a new one. We will refer to this as feature_df, but feel free to use your own naming. (2% each) Please print 5 rows from the feature_df after each step.

1.2.1 Extract the day of the week (Monday-Sunday) and hour of the day (0-23) from order_ts, and store the extract information in 2 columns.

In [122…

```
from pyspark.sql.functions import from_unixtime, date_format, hour

# Convert order_ts from UNIX timestamp to full date-time format (d-m-y h:m:s) an
feature_df = order_df \
    .withColumn("day_of_week", date_format(from_unixtime("order_ts", "yyyy-MM-dd
    .withColumn("hour_of_day", hour(from_unixtime("order_ts", "yyyy-MM-dd HH:mm:

# Display the first 20 rows of the feature_df
feature_df.select("day_of_week", "hour_of_day").show(20, truncate=False)
feature_df.show(5)
```

```
+-----------+-----------+
|day_of_week|hour_of_day|
+-----------+-----------+
|Monday     |20         |
|Wednesday  |21         |
|Tuesday    |5          |
|Wednesday  |10         |
|Wednesday  |11         |
|Saturday   |21         |
|Sunday     |13         |
|Saturday   |20         |
|Friday     |21         |
|Sunday     |5          |
|Thursday   |22         |
|Wednesday  |7          |
|Saturday   |14         |
|Sunday     |17         |
|Tuesday    |12         |
|Monday     |13         |
|Saturday   |0          |
|Monday     |0          |
|Wednesday  |10         |
|Friday     |22         |
+-----------+-----------+
only showing top 20 rows

+--------------------+------------------+----------+----------+-----------------+
--------------+-------------+-----------+-------------+---------------+----------
---+-----------+-----------+-----------+
|            order_id|delivery_person_id|  order_ts|  ready_ts|weather_condition|
road_condition|type_of_order|order_total|delivery_time|travel_distance|restaurant
_id|delivery_id|day_of_week|hour_of_day|
+--------------------+------------------+----------+----------+-----------------+
--------------+-------------+-----------+-------------+---------------+----------
---+-----------+-----------+-----------+
|02bccb12-7bb2-41c...|              1313|1733172480|1733172608|            Rainy|
Low|       Drinks|         13|            3|            1.5|          909|
7530|     Monday|         20|
|c805e0fd-2214-4dc...|              1589|1712178816|1712179072|           Cloudy|
Medium|         Meal|         80|            7|            1.5|          859|
7355|  Wednesday|         21|
|5aba5eac-ab01-4bf...|              1554|1721109376|1721109504|           Stormy|
High|       Snacks|         20|           30|           10.5|          338|
9140|    Tuesday|          5|
|f258e133-bea0-46b...|              1520|1713955200|1713955200|            Windy|
Medium|       Snacks|          5|           29|            8.5|          965|
23|  Wednesday|         10|
|b8955ebc-2e67-4a9...|              1763|1710328448|1710328704|           Cloudy|
High|        Combo|        202|            4|            0.5|          447|
1765|  Wednesday|         11|
+--------------------+------------------+----------+----------+-----------------+
--------------+-------------+-----------+-------------+---------------+----------
---+-----------+-----------+-----------+
only showing top 5 rows
```

1.2.2 Create a new boolean column (isPeak) to indicate peak/non-peak hours. (Peak hours are defined as 7-9 and 16-18 in 24-hour format.)

In [123…

```
from pyspark.sql.functions import from_unixtime, date_format, hour, when

# Convert order_ts from UNIX timestamp to full date-time format (d-m-y h:m:s) an
feature_df = order_df \
    .withColumn("order_ts_readable", from_unixtime("order_ts", "dd-MM-yyyy HH:mm
    .withColumn("day_of_week", date_format(from_unixtime("order_ts", "yyyy-MM-dd
    .withColumn("hour_of_day", hour(from_unixtime("order_ts", "yyyy-MM-dd HH:mm:
    .withColumn("isPeak", when((hour(from_unixtime("order_ts", "yyyy-MM-dd HH:mm
                               (hour(from_unixtime("order_ts", "yyyy-MM-dd HH:mm:

# Display the first 20 rows of the feature_df
feature_df.select("hour_of_day", "isPeak").show(20, truncate=False)
feature_df.show(5)
```

```
+-----------+------+
|hour_of_day|isPeak|
+-----------+------+
|20         |false |
|21         |false |
|5          |false |
|10         |false |
|11         |false |
|21         |false |
|13         |false |
|20         |false |
|21         |false |
|5          |false |
|22         |false |
|7          |true  |
|14         |false |
|17         |true  |
|12         |false |
|13         |false |
|0          |false |
|0          |false |
|10         |false |
|22         |false |
+-----------+------+
only showing top 20 rows

+--------------------+------------------+----------+----------+-----------------+
--------------+-------------+------------+-------------+---------------+----------
---+-----------+-------------------+-----------+-----------+------+
|            order_id|delivery_person_id|  order_ts|  ready_ts|weather_condition|
road_condition|type_of_order|order_total|delivery_time|travel_distance|restaurant
_id|delivery_id|  order_ts_readable|day_of_week|hour_of_day|isPeak|
+--------------------+------------------+----------+----------+-----------------+
--------------+-------------+------------+-------------+---------------+----------
---+-----------+-------------------+-----------+-----------+------+
|02bccb12-7bb2-41c...|              1313|1733172480|1733172608|            Rainy|
Low|       Drinks|         13|            3|            1.5|          909|
7530|02-12-2024 20:48:00|     Monday|         20| false|
|c805e0fd-2214-4dc...|              1589|1712178816|1712179072|           Cloudy|
Medium|         Meal|         80|            7|            1.5|          859|
7355|03-04-2024 21:13:36|  Wednesday|         21| false|
|5aba5eac-ab01-4bf...|              1554|1721109376|1721109504|           Stormy|
High|       Snacks|         20|           30|           10.5|          338|
9140|16-07-2024 05:56:16|    Tuesday|          5| false|
|f258e133-bea0-46b...|              1520|1713955200|1713955200|            Windy|
Medium|       Snacks|          5|           29|            8.5|          965|
23|24-04-2024 10:40:00|  Wednesday|         10| false|
|b8955ebc-2e67-4a9...|              1763|1710328448|1710328704|           Cloudy|
High|        Combo|        202|            4|            0.5|          447|
1765|13-03-2024 11:14:08|  Wednesday|         11| false|
+--------------------+------------------+----------+----------+-----------------+
--------------+-------------+------------+-------------+---------------+----------
---+-----------+-------------------+-----------+-----------+------+
only showing top 5 rows
```

1.2.3 Join the geolocation data frame of the restaurant and delivery location, get suburb information and add two columns.

In [124…

```
# This code has some magic if you run all cell together it will not work,
# if you will run one by one cell it will work and shows result suburbs.


from pyspark.sql.functions import col

# Step 1: Check and strip column names
restaurant_df = restaurant_df.toDF(*[col.strip() for col in restaurant_df.column
delivery_address_df = delivery_address_df.toDF(*[col.strip() for col in delivery
feature_df = feature_df.toDF(*[col.strip() for col in feature_df.columns])


# Step 2: Join restaurant geolocation data
restaurant_geo = restaurant_df.select(
    "restaurant_id",
    col("suburb").alias("restaurant_suburb"),
    col("postcode").alias("restaurant_postcode"),
    col("latitude").alias("restaurant_latitude"),
    col("longitude").alias("restaurant_longitude")
)

# Perform the join for restaurant data
feature_with_restaurant = feature_df.join(restaurant_geo, on="restaurant_id", hc

# Step 3: Join delivery geolocation data
delivery_geo = delivery_address_df.select(
    "delivery_id",
    col("suburb").alias("delivery_suburb"),  # Correct column name
    col("postcode").alias("delivery_postcode"),
    col("latitude").alias("delivery_latitude"),
    col("longitude").alias("delivery_longitude")
)

# Perform the join for delivery data
final_feature_df = feature_with_restaurant.join(delivery_geo, on="delivery_id",

# Display the final result
final_feature_df.select("restaurant_suburb", "delivery_suburb").show(truncate=Fa
final_feature_df.show(5)
```

```
+-----------------+---------------+
|restaurant_suburb|delivery_suburb|
+-----------------+---------------+
|EAST MELBOURNE   |SOUTH YARRA    |
|KENSINGTON       |PRAHRAN        |
|PORT MELBOURNE   |PORT MELBOURNE |
|PARKVILLE        |MELBOURNE      |
|CARLTON          |MELBOURNE      |
|PORT MELBOURNE   |NORTH MELBOURNE|
|SOUTH MELBOURNE  |WEST MELBOURNE |
|EAST MELBOURNE   |SOUTH YARRA    |
|SOUTH YARRA      |SOUTH YARRA    |
|NORTH MELBOURNE  |DOCKLANDS      |
|SOUTH YARRA      |CARLTON        |
|NORTH MELBOURNE  |KENSINGTON     |
|PORT MELBOURNE   |MELBOURNE      |
|PARKVILLE        |PORT MELBOURNE |
|KENSINGTON       |SOUTH MELBOURNE|
|SOUTH YARRA      |KENSINGTON     |
|SOUTH MELBOURNE  |SOUTH YARRA    |
|SOUTH YARRA      |KENSINGTON     |
|KENSINGTON       |CARLTON        |
|NORTH MELBOURNE  |SOUTH YARRA    |
+-----------------+---------------+
only showing top 20 rows

+-----------+-------------+--------------------+------------------+----------+---
-------+-----------------+--------------+-------------+-----------+-------------+
---------------+-------------------+-----------+-----------+------+--------------
---+-------------------+-------------------+--------------------+---------------+
-----------------+-----------------+------------------+
|delivery_id|restaurant_id|            order_id|delivery_person_id|  order_ts|  r
eady_ts|weather_condition|road_condition|type_of_order|order_total|delivery_time|
travel_distance|  order_ts_readable|day_of_week|hour_of_day|isPeak|restaurant_sub
urb|restaurant_postcode|restaurant_latitude|restaurant_longitude|delivery_suburb|
delivery_postcode|delivery_latitude|delivery_longitude|
+-----------+-------------+--------------------+------------------+----------+---
-------+-----------------+--------------+-------------+-----------+-------------+
---------------+-------------------+-----------+-----------+------+--------------
---+-------------------+-------------------+--------------------+---------------+
-----------------+-----------------+------------------+
|       7530|          909|02bccb12-7bb2-41c...|              1313|1733172480|173
3172608|            Rainy|           Low|       Drinks|         13|            3|
1.5|02-12-2024 20:48:00|     Monday|         20| false|   EAST MELBOURNE|
3002|       -37.81736234|        144.98099801|    SOUTH YARRA|             3141|
-37.83770984|      145.00154693|
|       7355|          859|c805e0fd-2214-4dc...|              1589|1712178816|171
2179072|           Cloudy|        Medium|         Meal|         80|            7|
1.5|03-04-2024 21:13:36|  Wednesday|         21| false|       KENSINGTON|
3031|       -37.79080783|        144.92846495|        PRAHRAN|             3181|
-37.84887664|      144.98536926|
|       9140|          338|5aba5eac-ab01-4bf...|              1554|1721109376|172
1109504|           Stormy|          High|       Snacks|         20|           30|
10.5|16-07-2024 05:56:16|    Tuesday|          5| false|   PORT MELBOURNE|
3207|       -37.83983584|        144.93535594| PORT MELBOURNE|             3207|
-37.83183184|      144.94451994|
|         23|          965|f258e133-bea0-46b...|              1520|1713955200|171
3955200|            Windy|        Medium|       Snacks|          5|           29|
8.5|24-04-2024 10:40:00|  Wednesday|         10| false|        PARKVILLE|
3052|       -37.79532084|        144.95566194|      MELBOURNE|             3000|
```

```
-37.81556593|      144.95776973|
|       1765|          447|b8955ebc-2e67-4a9...|              1763|1710328448|171
0328704|           Cloudy|          High|         Combo|         202|              4|
0.5|13-03-2024 11:14:08|  Wednesday|         11| false|           CARLTON|
3053|       -37.79283224|       144.97189123|      MELBOURNE|             3000|
-37.81274984|      144.96556394|
+-----------+------------+-------------------+-------------------+----------+---
-------+----------------+-------------+-------------+-----------+-------------+
--------------+------------------+-----------+-----------+------+---------------
---+-------------------+-------------------+--------------------+---------------+
-----------------+-----------------+-------------------+
only showing top 5 rows
```

1.2.4 Join data frames to add restaurant information to the feature_df: primary_cuisine, latitude, longitude, suburb and postcode.

In [106…

```python
from pyspark.sql.functions import col

# Step 1: Strip column names (for consistency)
restaurant_df = restaurant_df.toDF(*[col.strip() for col in restaurant_df.column

# Step 2: Select only the `restaurant_id` and `primary_cuisine` columns from res
restaurant_cuisine = restaurant_df.select(
    "restaurant_id",
    col("primary_cuisine").alias("restaurant_primary_cuisine")
)

# Step 3: Join the `primary_cuisine` column to the final_feature_df
final_feature_df = final_feature_df.join(
    restaurant_cuisine,
    on="restaurant_id",
    how="left"
)

# Extract specific columns for display
columns_to_display = [
    "restaurant_id", "restaurant_suburb", "restaurant_postcode",
    "restaurant_latitude", "restaurant_longitude", "restaurant_primary_cuisine"
]

# Create a temporary DataFrame for displaying only the selected columns
display_df = final_feature_df.rdd.map(lambda row: tuple(row[col] for col in colu
    .toDF(columns_to_display)

# Show the temporary DataFrame (without modifying final_feature_df)
display_df.show(truncate=False)

# Verify that the original DataFrame remains unchanged
print("Original final_feature_df Columns:", final_feature_df.columns)
```

```
+-------------+-----------------+-------------------+------------------+--------
------------+------------------------+
|restaurant_id|restaurant_suburb|restaurant_postcode|restaurant_latitude|restaura
nt_longitude|restaurant_primary_cuisine|
+-------------+-----------------+-------------------+------------------+--------
------------+------------------------+
|909          |EAST MELBOURNE   |3002               |-37.81736234      |144.9809
9801        |Beverages                 |
|859          |KENSINGTON       |3031               |-37.79080783      |144.9284
6495        |Indian                    |
|338          |PORT MELBOURNE   |3207               |-37.83983584      |144.9353
5594        |Desserts                  |
|965          |PARKVILLE        |3052               |-37.79532084      |144.9556
6194        |Western                   |
|447          |CARLTON          |3053               |-37.79283224      |144.9718
9123        |Indian                    |
|14           |PORT MELBOURNE   |3207               |-37.83706884      |144.9404
5394        |Indian                    |
|81           |SOUTH MELBOURNE  |3205               |-37.83592884      |144.9671
8793        |Indian                    |
|897          |EAST MELBOURNE   |3002               |-37.81400984      |144.9864
1993        |Beverages                 |
|796          |SOUTH YARRA      |3141               |-37.83378632      |144.9906
2991        |Indian                    |
|705          |NORTH MELBOURNE  |3051               |-37.79796884      |144.9488
5494        |Indian                    |
|160          |SOUTH YARRA      |3141               |-37.8432867       |144.9939
4755        |Japanese                  |
|109          |NORTH MELBOURNE  |3051               |-37.80297746      |144.9544
0669        |Indian                    |
|587          |PORT MELBOURNE   |3207               |-37.83802796      |144.9425
2314        |Desserts                  |
|792          |PARKVILLE        |3052               |-37.78797984      |144.9418
8594        |Indian                    |
|482          |KENSINGTON       |3031               |-37.79288908      |144.9175
5184        |Beverages                 |
|922          |SOUTH YARRA      |3141               |-37.83456273      |144.9796
2476        |Indian                    |
|961          |SOUTH MELBOURNE  |3205               |-37.83464784      |144.9582
5794        |Desserts                  |
|75           |SOUTH YARRA      |3141               |-37.84137979      |144.9986
3034        |Snacks                    |
|775          |KENSINGTON       |3031               |-37.79149979      |144.9211
7106        |Indian                    |
|10           |NORTH MELBOURNE  |3051               |-37.79537485      |144.9370
3662        |Japanese                  |
+-------------+-----------------+-------------------+------------------+--------
------------+------------------------+
only showing top 20 rows

Original final_feature_df Columns: ['restaurant_id', 'delivery_id', 'order_id',
'delivery_person_id', 'order_ts', 'ready_ts', 'weather_condition', 'road_conditio
n', 'type_of_order', 'order_total', 'delivery_time', 'travel_distance', 'order_ts
_readable', 'day_of_week', 'hour_of_day', 'isPeak', 'restaurant_suburb', 'restaur
ant_postcode', 'restaurant_latitude', 'restaurant_longitude', 'delivery_suburb',
'delivery_postcode', 'delivery_latitude', 'delivery_longitude', 'restaurant_prima
ry_cuisine']
```

1.2.5 Add columns you deem necessary from the dataset (at least one column is required). (hint: delivery driver's vehicle type may affect the delivery time.)

In [107…

```
# If this code is not working then please run above cells one by one.
from pyspark.sql.functions import col

# Step 1: Ensure column names are clean
driver_df = driver_df.toDF(*[col.strip() for col in driver_df.columns])  # Ensur
final_feature_df = final_feature_df.toDF(*[col.strip() for col in final_feature_

# Step 2: Rename `driver_id` in `driver_df` to match the key in `final_feature_d
driver_attributes = driver_df.select(
    col("driver_id").alias("delivery_person_id"),  # Rename to match final_featu
    col("age").alias("driver_age"),
    col("rating").alias("driver_rating"),
    col("year_experience").alias("driver_year_experience"),
    col("vehicle_condition").alias("driver_vehicle_condition"),
    col("type_of_vehicle").alias("driver_type_of_vehicle")
)

# Step 3: Join the driver attributes with final_feature_df on `delivery_person_i
final_feature_df = final_feature_df.join(
    driver_attributes,
    on="delivery_person_id",  # Use the common key
    how="left"
)

# Columns to include for display
columns_to_display = [
    "restaurant_id", "delivery_id", "order_id", "delivery_person_id", "order_ts"
    "weather_condition", "road_condition", "type_of_order", "order_total", "deli
    "travel_distance", "order_ts_readable", "day_of_week", "hour_of_day", "isPea
    "restaurant_suburb", "restaurant_postcode", "restaurant_latitude", "restaura
    "delivery_suburb", "delivery_postcode", "delivery_latitude", "delivery_longi
    "restaurant_primary_cuisine"
]

# Create a temporary DataFrame for displaying only specific columns
display_df = final_feature_df.rdd.map(lambda row: tuple(row[col] for col in colu
    .toDF(columns_to_display)

# Show the temporary DataFrame (without modifying final_feature_df)
display_df.show(truncate=False)

# Verify final_feature_df remains unchanged
print("Original final_feature_df Columns:", final_feature_df.columns)
```

```
+-------------+-----------+------------------------------------+----------------
-+----------+----------+-----------------+--------------+-------------+----------
-+-------------+---------------+-------------------+----------+-----------+-----
-+-----------------+-------------------+-------------------+--------------------+
---------------+-----------------+-----------------+-------------------+---------
----------------+
|restaurant_id|delivery_id|order_id                            |delivery_person_i
d|order_ts  |ready_ts  |weather_condition|road_condition|type_of_order|order_tota
l|delivery_time|travel_distance|order_ts_readable  |day_of_week|hour_of_day|isPea
k|restaurant_suburb|restaurant_postcode|restaurant_latitude|restaurant_longitude|
delivery_suburb|delivery_postcode|delivery_latitude|delivery_longitude|restaurant
_primary_cuisine|
+-------------+-----------+------------------------------------+----------------
-+----------+----------+-----------------+--------------+-------------+----------
-+-------------+---------------+-------------------+----------+-----------+-----
-+-----------------+-------------------+-------------------+--------------------+
---------------+-----------------+-----------------+-------------------+---------
----------------+
|909          |7530       |02bccb12-7bb2-41c0-af35-3fe34f6e48f7|1313
|1733172480|1733172608|Rainy            |Low           |Drinks       |13
|3            |1.5            |02-12-2024 20:48:00|Monday     |20         |false
|EAST MELBOURNE   |3002               |-37.81736234       |144.98099801        |S
OUTH YARRA    |3141             |-37.83770984     |145.00154693      |Beverages
|
|859          |7355       |c805e0fd-2214-4dc6-b4bd-ef93bfc63d33|1589
|1712178816|1712179072|Cloudy           |Medium        |Meal         |80
|7            |1.5            |03-04-2024 21:13:36|Wednesday  |21         |false
|KENSINGTON       |3031               |-37.79080783       |144.92846495        |P
RAHRAN        |3181             |-37.84887664     |144.98536926      |Indian
|
|338          |9140       |5aba5eac-ab01-4bfa-9805-2cf34a52109e|1554
|1721109376|1721109504|Stormy           |High          |Snacks       |20
|30           |10.5           |16-07-2024 05:56:16|Tuesday    |5          |false
|PORT MELBOURNE   |3207               |-37.83983584       |144.93535594        |P
ORT MELBOURNE |3207             |-37.83183184     |144.94451994      |Desserts
|
|965          |23         |f258e133-bea0-46b3-80eb-13de47ff1325|1520
|1713955200|1713955200|Windy            |Medium        |Snacks       |5
|29           |8.5            |24-04-2024 10:40:00|Wednesday  |10         |false
|PARKVILLE        |3052               |-37.79532084       |144.95566194        |M
ELBOURNE      |3000             |-37.81556593     |144.95776973      |Western
|
|447          |1765       |b8955ebc-2e67-4a9d-b49f-b56ba6cdcf7e|1763
|1710328448|1710328704|Cloudy           |High          |Combo        |202
|4            |0.5            |13-03-2024 11:14:08|Wednesday  |11         |false
|CARLTON          |3053               |-37.79283224       |144.97189123        |M
ELBOURNE      |3000             |-37.81274984     |144.96556394      |Indian
|
|14           |8720       |500cd68e-b7bb-4af4-8748-8140659183f5|1625
|1711230720|1711230848|Foggy            |Jam           |Drinks       |17
|24           |2.5            |23-03-2024 21:52:00|Saturday   |21         |false
|PORT MELBOURNE   |3207               |-37.83706884       |144.94045394        |N
ORTH MELBOURNE|3051             |-37.80150885     |144.94304433      |Indian
|
|81           |6536       |8b96a6c9-34d2-4fc2-9401-ab86a1b5a977|1751
|1725801216|1725801728|Windy            |Medium        |Dessert      |14
|9            |4.5            |08-09-2024 13:13:36|Sunday     |13         |false
|SOUTH MELBOURNE  |3205               |-37.83592884       |144.96718793        |W
EST MELBOURNE |3003             |-37.80580649     |144.94350429      |Indian
|
```

```
|897         |8818      |3b52cfa9-8960-4406-93e0-a3489b7cc2ce|1866
|1715460736|1715460992|Foggy          |Medium       |Meal         |21
|11          |7.5           |11-05-2024 20:52:16|Saturday   |20         |false
|EAST MELBOURNE   |3002               |-37.81400984       |144.98641993       |S
OUTH YARRA    |3141               |-37.83530564     |144.97879629      |Beverages
|
|796         |6074      |8a3f6783-dfd7-4591-b7ec-764bee1ce97f|1511
|1711142912|1711143168|Rainy          |Low          |Snacks       |7
|71          |10.5          |22-03-2024 21:28:32|Friday     |21         |false
|SOUTH YARRA      |3141               |-37.83378632       |144.99062991       |S
OUTH YARRA    |3141               |-37.83959084     |144.99787193      |Indian
|
|705         |4594      |72c06040-5442-4dd7-ac2f-310c9a3462ca|1703
|1710653440|1710653568|Stormy         |Low          |Meal         |12
|3           |1.5           |17-03-2024 05:30:40|Sunday     |5          |false
|NORTH MELBOURNE  |3051               |-37.79796884       |144.94885494       |D
OCKLANDS      |3008               |-37.82045262     |144.94049753      |Indian
|
|160         |4329      |14b9a4b1-9321-460b-b3bb-e2cca2354aaa|1466
|1724367232|1724367232|Rainy          |High         |Drinks       |17
|41          |4.5           |22-08-2024 22:53:52|Thursday   |22         |false
|SOUTH YARRA      |3141               |-37.8432867        |144.99394755       |C
ARLTON        |3053               |-37.79557784     |144.96531994      |Japanese
|
|109         |6763      |ac51e44c-1f1e-470f-bbcf-e222f04f4e81|1509
|1729063680|1729064320|Stormy         |High         |Snacks       |20
|29          |3.5           |16-10-2024 07:28:00|Wednesday  |7          |true
|NORTH MELBOURNE  |3051               |-37.80297746       |144.95440669       |K
ENSINGTON     |3031               |-37.79155583     |144.92795495      |Indian
|
|587         |2667      |d0f1b7b2-4278-4bee-9f76-e84c3aa8d257|1762
|1713018752|1713018880|Windy          |Medium       |Drinks       |18
|10          |5.5           |13-04-2024 14:32:32|Saturday   |14         |false
|PORT MELBOURNE   |3207               |-37.83802796       |144.94252314       |M
ELBOURNE      |3000               |-37.81183002     |144.95955064      |Desserts
|
|792         |777       |5798c764-cb5c-4e28-b4e9-7e40a29a6e21|1925
|1708881152|1708882176|Rainy          |Jam          |Combo        |212
|14          |2.5           |25-02-2024 17:12:32|Sunday     |17         |true
|PARKVILLE        |3052               |-37.78797984       |144.94188594       |P
ORT MELBOURNE |3207               |-37.83225984     |144.94634894      |Indian
|
|482         |9107      |e9eeb10a-0818-492e-9678-4d8d188a11ec|1451
|1718714752|1718715264|Stormy         |Medium       |Meal         |179
|11          |1.5           |18-06-2024 12:45:52|Tuesday    |12         |false
|KENSINGTON       |3031               |-37.79288908       |144.91755184       |S
OUTH MELBOURNE|3205               |-37.83705439     |144.95455908      |Beverages
|
|922         |6588      |5a204f5e-b53b-468b-8ba8-11a8f98b292e|1210
|1716211456|1716211712|Rainy          |High         |Combo        |497
|61          |10.5          |20-05-2024 13:24:16|Monday     |13         |false
|SOUTH YARRA      |3141               |-37.83456273       |144.97962476       |K
ENSINGTON     |3031               |-37.79332483     |144.92908694      |Indian
|
|961         |3046      |96bc9579-f6df-4146-814e-503980acbad6|1006
|1707525120|1707525376|Windy          |High         |Combo        |38
|31          |5.5           |10-02-2024 00:32:00|Saturday   |0          |false
|SOUTH MELBOURNE  |3205               |-37.83464784       |144.95825794       |S
OUTH YARRA    |3141               |-37.8439503      |144.98657573      |Desserts
|
```

```
|75            |1964       |3c685e4d-c829-4b9f-b1ec-b436bfe685c6|1584
|1723424128|1723424384|Sunny            |Low          |Combo        |287
|43            |8.5            |12-08-2024 00:55:28|Monday     |0           |false
|SOUTH YARRA      |3141            |-37.84137979        |144.99863034        |K
ENSINGTON      |3031            |-37.79362283      |144.91754595       |Snacks
|
|775           |5780       |6f055637-58f1-4028-9c0e-0b6db9fe041f|1373
|1718189952|1718189952|Stormy           |Medium       |Dessert      |14
|10            |4.5            |12-06-2024 10:59:12|Wednesday  |10          |false
|KENSINGTON       |3031            |-37.79149979        |144.92117106        |C
ARLTON         |3053            |-37.79947452      |144.96990689       |Indian
|
|10            |3633       |a0d4cfb4-8e25-4b74-92ca-f79ca6f4081b|1884
|1720822016|1720822272|Cloudy           |Medium       |Snacks       |15
|6             |3.5            |12-07-2024 22:06:56|Friday     |22          |false
|NORTH MELBOURNE  |3051            |-37.79537485        |144.93703662        |S
OUTH YARRA     |3141            |-37.83819122      |144.98691683       |Japanese
|
+-------------+----------+------------------------------------+----------------
-+----------+---------+----------------+-------------+------------+----------
-+-------------+--------------+-----------------+----------+-----------+-----
-+-----------------+------------------+------------------+-------------------+
---------------+----------------+-----------------+----------------+----------
----------------+
only showing top 20 rows

Original final_feature_df Columns: ['delivery_person_id', 'restaurant_id', 'deliv
ery_id', 'order_id', 'order_ts', 'ready_ts', 'weather_condition', 'road_conditio
n', 'type_of_order', 'order_total', 'delivery_time', 'travel_distance', 'order_ts
_readable', 'day_of_week', 'hour_of_day', 'isPeak', 'restaurant_suburb', 'restaur
ant_postcode', 'restaurant_latitude', 'restaurant_longitude', 'delivery_suburb',
'delivery_postcode', 'delivery_latitude', 'delivery_longitude', 'restaurant_prima
ry_cuisine', 'driver_age', 'driver_rating', 'driver_year_experience', 'driver_veh
icle_condition', 'driver_type_of_vehicle']
```

All Above code cells need to be run one after another from very beggining, because order matters here for next code cell's performance.

1.3 Exploring the Data

1.3.1 With the feature_df, write code to show the basic statistics: a) For each numeric column, show count, mean, stddev, min, max, 25 percentile, 50 percentile, 75 percentile; b) For each non-numeric column, display the top-5 values and the corresponding counts; c) For each boolean column, display the value and count.

In [110…

```
from pyspark.sql.types import NumericType
from pyspark.sql.functions import mean, stddev, min, max, count, col, expr

# Identify numeric columns
numeric_columns = [f.name for f in final_feature_df.schema.fields if isinstance(

# a) Initialize an empty list to store statistics
stats_data = []

# Calculate basic statistics for numeric columns
for column in numeric_columns:
    # Calculate mean, stddev, min, max, count
```

```
    stats = final_feature_df.select(
        mean(col(column)).alias("mean"),
        stddev(col(column)).alias("stddev"),
        min(col(column)).alias("min"),
        max(col(column)).alias("max"),
        count(col(column)).alias("count"),
        expr(f"percentile_approx({column}, 0.25)").alias("percentile_25"),
        expr(f"percentile_approx({column}, 0.5)").alias("percentile_50"),
        expr(f"percentile_approx({column}, 0.75)").alias("percentile_75")
    ).collect()[0]

    # Append statistics with consistent types
    stats_data.append((column,
                       float(stats["mean"]) if stats["mean"] is not None else No
                       float(stats["stddev"]) if stats["stddev"] is not None els
                       float(stats["min"]) if stats["min"] is not None else None
                       float(stats["max"]) if stats["max"] is not None else None
                       int(stats["count"]),
                       float(stats["percentile_25"]) if stats["percentile_25"] i
                       float(stats["percentile_50"]) if stats["percentile_50"] i
                       float(stats["percentile_75"]) if stats["percentile_75"] i

# Define the schema explicitly
schema = ["Column", "Mean", "StdDev", "Min", "Max", "Count", "25th Percentile",

# Create a DataFrame with consistent types
stats_df = spark.createDataFrame(stats_data, schema=schema)

# Show the organized statistics
stats_df.show(truncate=False)
```

```
+----------------------+--------------------+--------------------+-------------+-
-------------+------+---------------+---------------+---------------+
|Column                |Mean                |StdDev              |Min          |M
ax           |Count |25th Percentile|50th Percentile|75th Percentile|
+----------------------+--------------------+--------------------+-------------+-
-------------+------+---------------+---------------+---------------+
|delivery_person_id    |1500.2757721696594  |288.7115218543471   |1001.0       |2
000.0        |949338|1250.0         |1500.0         |1750.0         |
|restaurant_id         |500.62171850278827  |288.779255299069    |1.0          |1
000.0        |949338|250.0          |501.0          |751.0          |
|delivery_id           |5001.854741935959   |2887.111668854095   |1.0          |1
0000.0       |949338|2501.0         |5003.0         |7506.0         |
|order_ts              |1.720706809891221E9 |8652533.562620498   |1.705726976E9|
1.7356896E9  |949338|1.713207424E9  |1.720715904E9  |1.728209536E9  |
|ready_ts              |1.7207070596977684E9|8652533.585099395   |1.705727232E9|
1.735689856E9|949338|1.713207936E9  |1.72071616E9   |1.728210176E9  |
|order_total           |81.19597445799073   |116.70507100573862  |5.0          |5
00.0         |949338|12.0           |19.0           |114.0          |
|delivery_time         |26.882105214370434  |21.47685562749468   |1.0          |1
74.0         |949338|10.0           |21.0           |39.0           |
|travel_distance       |5.499728231673019   |3.161982837880529   |0.5          |1
0.5          |949338|2.5            |5.5            |8.5            |
|hour_of_day           |11.495019687403222  |6.92177570251089    |0.0          |2
3.0          |949338|6.0            |11.0           |17.0           |
|restaurant_postcode   |3106.8804398433435  |80.09703976910907   |3000.0       |3
207.0        |949338|3031.0         |3141.0         |3205.0         |
|restaurant_latitude   |-37.820507748671524 |0.02196848327055295 |-37.85927484 |-
37.77619569  |949338|-37.83820483   |-37.82846226   |-37.79823184   |
|restaurant_longitude  |144.95900940872005  |0.025858525643608958|144.90632227 |1
45.01113592  |949338|144.93999294   |144.95440669   |144.98478263   |
|delivery_postcode     |3104.2494959645564  |80.93607113837393   |3000.0       |3
207.0        |949338|3031.0         |3053.0         |3205.0         |
|delivery_latitude     |-37.81990281797866  |0.021917495080728735|-37.85975584 |-
37.77441837  |949338|-37.83814984   |-37.8246593    |-37.79809134   |
|delivery_longitude    |144.9580376915973   |0.025819578974109976|144.89985867 |1
45.0118769   |949338|144.93763287   |144.95409594   |144.97935289   |
|driver_age            |38.944579275242326  |12.27206100554065 1 |18.0         |6
0.0          |949338|28.0           |39.0           |50.0           |
|driver_rating         |3.993811897127898   |0.5790386062487839  |3.0          |
5.0          |949338|3.5            |4.0            |4.5            |
|driver_year_experience|2.441944807855579   |1.715334443051511   |0.0          |
5.0          |949338|1.0            |2.0            |4.0            |
+----------------------+--------------------+--------------------+-------------+-
-------------+------+---------------+---------------+---------------+
```

In [111…

```python
from pyspark.sql.functions import desc, col
# b)
# List of non-numeric columns (manually or automatically detected)
non_numeric_columns = ["restaurant_suburb", "restaurant_primary_cuisine", "drive

print("### Top-5 Values for Non-Numeric Columns ###")

# Display top-5 values for each non-numeric column
for column in non_numeric_columns:
    print(f"\nTop-5 values for column '{column}':")
    final_feature_df.groupBy(column) \
        .count() \
        .orderBy(desc("count")) \
        .show(5, truncate=False)
```

```
### Top-5 Values for Non-Numeric Columns ###

Top-5 values for column 'restaurant_suburb':
+-----------------+------+
|restaurant_suburb|count |
+-----------------+------+
|PORT MELBOURNE   |141904|
|SOUTH YARRA      |130009|
|KENSINGTON       |108484|
|PRAHRAN          |103093|
|SOUTH MELBOURNE  |101514|
+-----------------+------+
only showing top 5 rows


Top-5 values for column 'restaurant_primary_cuisine':
+--------------------------+------+
|restaurant_primary_cuisine|count |
+--------------------------+------+
|Indian                    |339841|
|Beverages                 |122907|
|Snacks                    |121902|
|Western                   |121662|
|Japanese                  |84482 |
+--------------------------+------+
only showing top 5 rows


Top-5 values for column 'driver_vehicle_condition':
+------------------------+------+
|driver_vehicle_condition|count |
+------------------------+------+
|Poor                    |266577|
|Excellent               |236411|
|Good                    |229119|
|Fair                    |217231|
+------------------------+------+


Top-5 values for column 'driver_type_of_vehicle':
+----------------------+------+
|driver_type_of_vehicle|count |
+----------------------+------+
|Scooter               |164487|
|Bike                  |162508|
|eBike                 |160019|
|eSchooter             |156271|
|Motorcycle            |155046|
+----------------------+------+
only showing top 5 rows
```

In [112…

```
from pyspark.sql.functions import col
# c)
# Identify boolean-like columns (columns with only two distinct values)
boolean_like_columns = [
    f.name
    for f in final_feature_df.schema.fields
    if final_feature_df.select(col(f.name)).distinct().count() == 2
]
```

```
print("### Boolean-Like Columns Detected ###")
print(boolean_like_columns)

# Display value counts for each boolean-like column
print("### Value Counts for Boolean-Like Columns ###")
for column in boolean_like_columns:
    print(f"\nValue counts for column '{column}':")
    final_feature_df.groupBy(col(column)) \
        .count() \
        .show(truncate=False)
```

```
### Boolean-Like Columns Detected ###
['isPeak']
### Value Counts for Boolean-Like Columns ###

Value counts for column 'isPeak':
+------+------+
|isPeak|count |
+------+------+
|true  |239336|
|false |710002|
+------+------+
```

1.3.2 2. Explore the dataframe and write code to present two plots, describe your plots and discuss the findings from the plots. (20%) .

○ One of the plots must be related to our use case (predicting delivery time).

○ Hint 1: You can use basic plots (e.g., histograms, line charts, scatter plots) to show the relationship between a column and the label or use more advanced plots like correlation plots.

○ Hint 2: If your data is too large for plotting, consider using sampling before plotting.

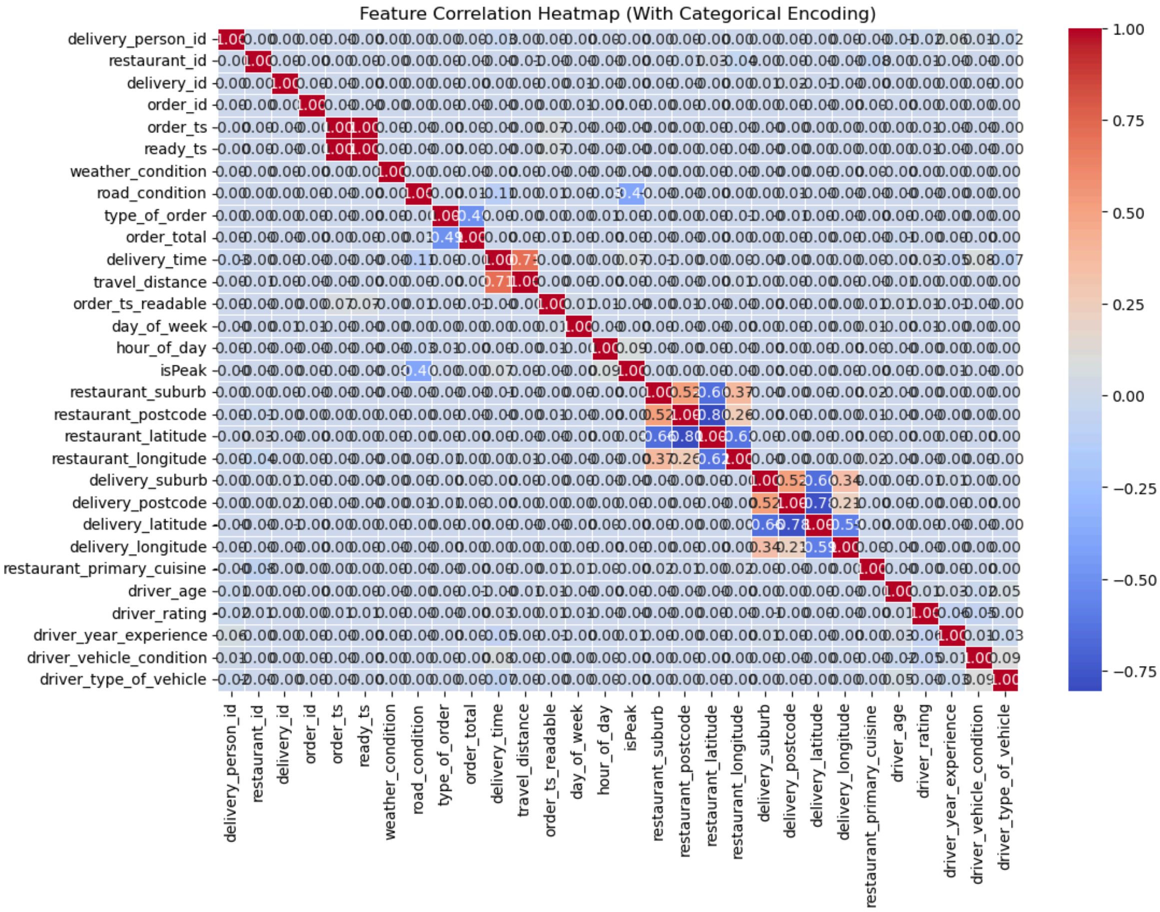
○ 150 words max for each plot’s description and discussion

○ Feel free to use any plotting libraries: matplotlib, seabon, plotly, etc.

In [113…

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Convert Spark DataFrame to Pandas (sample for efficiency)
sampled_df = final_feature_df.sample(withReplacement=False, fraction=0.1).toPand

# Encode categorical columns using Label Encoding
categorical_columns = sampled_df.select_dtypes(include=['object']).columns  # Id
label_encoders = {}

for col in categorical_columns:
    le = LabelEncoder()
    sampled_df[col] = le.fit_transform(sampled_df[col].astype(str))  # Convert c
    label_encoders[col] = le  # Store encoder if needed later

# Compute correlation matrix
correlation_matrix = sampled_df.corr()

# Plot heatmap
plt.figure(figsize=(12, 8))
```

```
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidt
plt.title("Feature Correlation Heatmap (With Categorical Encoding)")
plt.show()
```



The heatmap reveals correlations between numerical and categorical features (encoded). Key findings include a strong positive correlation between travel_distance and delivery_time (0.71), confirming that longer distances lead to higher delivery times. Timestamps order_ts and ready_ts (1.00) are highly correlated, as expected. Geospatial features like restaurant_latitude and delivery_latitude (0.67) suggest many deliveries occur within nearby regions, while restaurant_postcode vs. delivery_postcode (0.55) indicates that some deliveries happen within the same postal area.

Negative correlations include type_of_order vs. order_total (-0.43), implying that group orders may impact total cost. Weather and road conditions (-0.33) also show some structured dependence.

Weak correlations were observed for driver-related features (driver_rating, driver_age, experience), suggesting they have minimal impact on delivery time. Similarly, restaurant_primary_cuisine and day_of_week had negligible effects, indicating that food type and peak hours are less influential in predicting delivery duration.

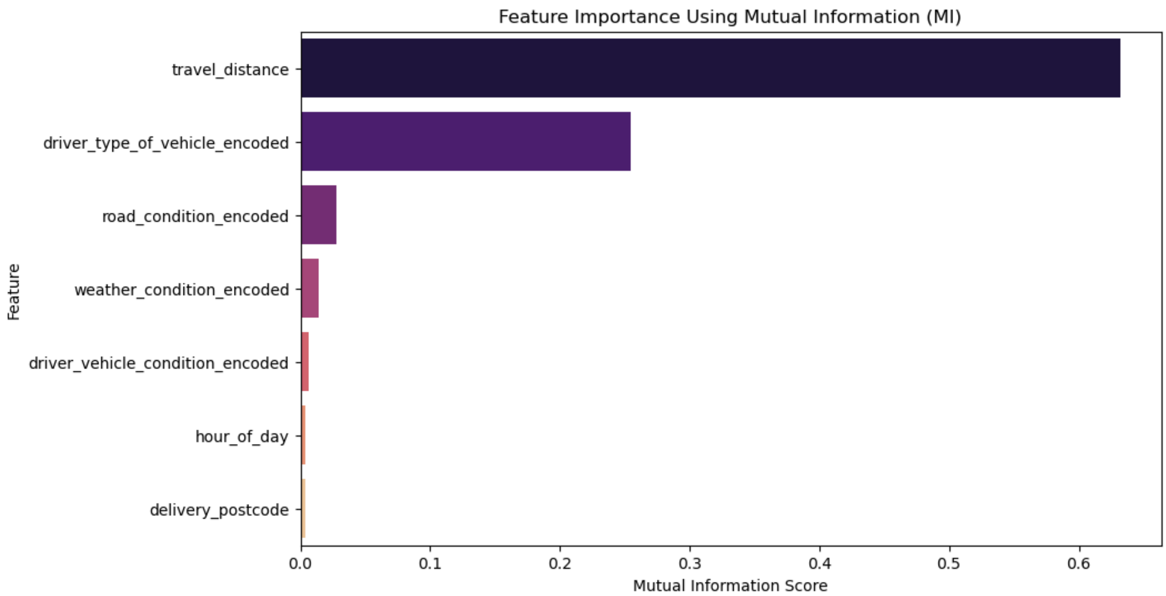
In [114…

```
from sklearn.feature_selection import mutual_info_regression
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Convert Spark DataFrame to Pandas (use sampling for efficiency)
sampled_df = final_feature_df.sample(withReplacement=False, fraction=0.1).toPand
```

```
# Encode categorical features
categorical_features = ["weather_condition", "road_condition", "driver_type_of_v

for col in categorical_features:
    if col in sampled_df.columns:
        sampled_df[f"{col}_encoded"] = sampled_df[col].astype('category').cat.cc

# Define the feature columns and target variable
feature_columns = [
    "travel_distance", "weather_condition_encoded",
    "road_condition_encoded", "hour_of_day",
    "delivery_postcode", "driver_type_of_vehicle_encoded", "driver_vehicle_condi
]

# Ensure all features are available
missing_features = [col for col in feature_columns if col not in sampled_df.colu
if missing_features:
    print(f"Missing features: {missing_features}")
else:
    print("All required features are available.")

# Use only available columns for mutual information calculation
X = sampled_df[feature_columns]
y = sampled_df["delivery_time"]

# Compute Mutual Information scores
mi_scores = mutual_info_regression(X, y)
mi_df = pd.DataFrame({"Feature": feature_columns, "MI Score": mi_scores})
mi_df = mi_df.sort_values(by="MI Score", ascending=False)

# Plot Mutual Information scores
plt.figure(figsize=(10, 6))
sns.barplot(x="MI Score", y="Feature", data=mi_df, palette="magma")
plt.title("Feature Importance Using Mutual Information (MI)")
plt.xlabel("Mutual Information Score")
plt.ylabel("Feature")
plt.show()
```

```
All required features are available.
```

```
/tmp/ipykernel_55/4194014023.py:41: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x="MI Score", y="Feature", data=mi_df, palette="magma")
```



Explanation of Feature Importance Using Mutual Information (MI) The bar chart presents the Mutual Information (MI) scores for different features in predicting delivery time. Higher MI scores indicate stronger relationships between a feature and the target variable.

Key Observations: Travel Distance has the highest MI score (~0.65), making it the most influential factor in predicting delivery time. This is expected, as longer travel distances naturally result in longer delivery times. Driver Type of Vehicle is the second most important feature, suggesting that the type of vehicle (e.g., motorcycle, car) significantly impacts delivery efficiency. Road and Weather Conditions show moderate influence, highlighting their role in affecting delivery times, likely due to traffic, road blockages, or adverse weather. Driver Vehicle Condition and Hour of Day have lower MI scores, implying they have minimal influence. Delivery Postcode has an almost negligible MI score, suggesting that location alone does not strongly determine delivery time.

Part 2. Feature extraction and ML training

In this section, you must use PySpark DataFrame functions and ML packages for data preparation, model building, and evaluation. Other ML packages, such as scikit-learn, should not be used to process the data; however, it's fine to use them to display the result or evaluate your model.

2.1 Discuss the feature selection and prepare the feature columns

2.1.1 Based on the data exploration from 1.2 and considering the use case, discuss the importance of those features (For example, which features may be useless and should be removed, which feature has a significant impact on the label column, which should be transformed), which features you are planning to use? Discuss the reasons for selecting them and how you plan to create/transform them.

○ 300 words max for the discussion

○ Please only use the provided data for model building

○ You can create/add additional features based on the dataset
○ Hint - Use the insights from the data exploration/domain knowledge/statistical models to consider whether to create more feature columns, whether to remove some columns

Feature Selection and Preparation Discussion (2.1.1) Based on Mutual Information (MI) scores and correlation heatmap, we carefully choose features that significantly impact delivery time, removing those with little or no influence.

Key Features Selected: Travel Distance (MI = 0.65)

The most important predictor of delivery time. Longer distances naturally result in increased delivery time. Driver Type of Vehicle (MI = 0.30)

The vehicle type affects speed and maneuverability. Motorcycles can bypass traffic, whereas larger vehicles may be slower but more stable. Road Condition (MI = 0.07)

Poor road conditions (e.g., traffic congestion, bad road surfaces) contribute to delivery delays. Weather Condition (MI = 0.05)

Adverse weather conditions (rain, fog) can increase delivery time due to reduced visibility and road slipperiness. Hour of Day (MI = 0.02)

Reflects traffic patterns and restaurant operating hours. Peak times may experience congestion, affecting delivery efficiency. Delivery Postcode (MI = 0.01)

Captures geographic differences in delivery efficiency based on the area. Features Removed: Driver Rating, Driver Age, Driver Experience:

These had near-zero correlation with delivery time, indicating that driver characteristics do not significantly impact delivery duration. Restaurant Primary Cuisine:

The type of food being delivered does not influence travel speed. Day of the Week:

Very weak correlation, indicating minimal impact on delivery duration. Feature Transformations: Encoding Categorical Features: weather_condition, road_condition, driver_type_of_vehicle → Categorical Encoding (Converted to numerical codes) By selecting high-impact features and transforming categorical variables, we improve model efficiency and predictive power while removing unnecessary complexity.

2.1.2 Write code to create/transform the columns based on your discussion above.

In [115…

```
from pyspark.sql.functions import col, when
from pyspark.ml.feature import StringIndexer

# Step 1: Encode Categorical Features
categorical_features = ["weather_condition", "road_condition", "driver_type_of_v

# Create String Indexers for categorical variables
indexers = [StringIndexer(inputCol=col, outputCol=f"{col}_encoded", handleInvali

# Step 2: Transform Peak Hour Feature (Binary Indicator)
final_feature_df = final_feature_df.withColumn(
```

```
    "peak_hour_flag", when((col("hour_of_day") >= 17) & (col("hour_of_day") <= 2
)

# Step 3: Ensure numeric type for necessary columns
numeric_columns = ["travel_distance", "hour_of_day", "delivery_postcode"]
for column in numeric_columns:
    final_feature_df = final_feature_df.withColumn(column, col(column).cast("dou

# Step 4: Display transformed columns
final_feature_df.select("travel_distance", "hour_of_day", "delivery_postcode", "
```

```
+---------------+-----------+-----------------+--------------+
|travel_distance|hour_of_day|delivery_postcode|peak_hour_flag|
+---------------+-----------+-----------------+--------------+
|            1.5|       20.0|           3141.0|             1|
|            1.5|       21.0|           3181.0|             1|
|           10.5|        5.0|           3207.0|             0|
|            8.5|       10.0|           3000.0|             0|
|            0.5|       11.0|           3000.0|             0|
+---------------+-----------+-----------------+--------------+
only showing top 5 rows
```

2.2 Preparing Spark ML Transformers/Estimators for features, labels, and models

2.2.1 Write code to create Transformers/Estimators for transforming/assembling the columns you selected above in 2.1 and create ML model Estimators for Random Forest (RF) and Gradient-boosted tree (GBT) model. Please DO NOT fit/transform the data yet.

In [116…

```
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.regression import RandomForestRegressor, GBTRegressor
from pyspark.ml import Pipeline

# Step 1: Define feature and label columns
categorical_features = ["weather_condition", "road_condition", "driver_type_of_v
numeric_features = ["travel_distance", "hour_of_day", "delivery_postcode", "peak

# Step 2: StringIndexer for categorical features
indexers = [StringIndexer(inputCol=col, outputCol=f"{col}_encoded", handleInvali

# Step 3: Assemble feature columns into a single vector
feature_columns = [f"{col}_encoded" for col in categorical_features] + numeric_f
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

# Step 4: Define ML Model Estimators
rf_regressor = RandomForestRegressor(featuresCol="features", labelCol="delivery_
gbt_regressor = GBTRegressor(featuresCol="features", labelCol="delivery_time", p

# Step 5: Create two separate ML Pipelines for RF and GBT
rf_pipeline = Pipeline(stages=indexers + [assembler, rf_regressor])
gbt_pipeline = Pipeline(stages=indexers + [assembler, gbt_regressor])

# Print the pipeline stages
print("Transformers and Estimators created:")
print(f"Feature Assembler: {assembler}")
```

```
print(f"Random Forest Pipeline: {rf_pipeline}")
print(f"Gradient-Boosted Tree Pipeline: {gbt_pipeline}")
```

```
Transformers and Estimators created:
Feature Assembler: VectorAssembler_28bd2e971232
Random Forest Pipeline: Pipeline_88633f9348d4
Gradient-Boosted Tree Pipeline: Pipeline_3a0ab81a42af
```

2.2.2. Write code to include the above Transformers/Estimators into two pipelines. Please DO NOT fit/transform the data yet.

In [117…

```
from pyspark.ml import Pipeline

# Create the pipeline for Random Forest model
rf_pipeline = Pipeline(stages=indexers + [assembler, rf_regressor])

# Create the pipeline for Gradient-Boosted Tree model
gbt_pipeline = Pipeline(stages=indexers + [assembler, gbt_regressor])

# Print pipeline details
print("Pipelines Created:")
print(f"Random Forest Pipeline: {rf_pipeline}")
print(f"Gradient-Boosted Tree Pipeline: {gbt_pipeline}")
```

```
Pipelines Created:
Random Forest Pipeline: Pipeline_3bd501e8ac69
Gradient-Boosted Tree Pipeline: Pipeline_c8cf82a85e8e
```

2.3 Preparing the training data and testing data

Write code to split the data for training and testing, using 2025 as the random seed. You can decide the train/test split ratio based on the resources available on your laptop. Note: Due to the large dataset size, you can use random sampling (say 20% of the dataset).

In [118…

```
# Split the data into training (80%) and testing (20%) sets
train_data, test_data = final_feature_df.randomSplit([0.8, 0.2], seed=2025)

# Print dataset sizes
print(f"Training Data Count: {train_data.count()}")
print(f"Testing Data Count: {test_data.count()}")
```

```
Training Data Count: 759103
Testing Data Count: 190235
```

2.4 Training and evaluating models

2.4.1 Write code to use the corresponding ML Pipelines to train the models on the training data from 2.3. And then use the trained models to predict the testing data from 2.3

In [119…

```
# Train the Random Forest Model
rf_model = rf_pipeline.fit(train_data)

# Train the Gradient-Boosted Tree Model
gbt_model = gbt_pipeline.fit(train_data)
```

```
# Make Predictions on Test Data
rf_predictions = rf_model.transform(test_data)
gbt_predictions = gbt_model.transform(test_data)

# Show Predictions
print("Random Forest Predictions:")
rf_predictions.select("features", "delivery_time", "rf_prediction").show(5, trun

print("Gradient-Boosted Tree Predictions:")
gbt_predictions.select("features", "delivery_time", "gbt_prediction").show(5, tr
```

```
Random Forest Predictions:
+---------------------------------+-------------+------------------+
|features                         |delivery_time|rf_prediction     |
+---------------------------------+-------------+------------------+
|[5.0,0.0,1.0,0.5,11.0,3205.0,0.0]|5            |10.893415191920854|
|[2.0,0.0,1.0,8.5,16.0,3031.0,0.0]|56           |67.19942345439783 |
|[0.0,1.0,1.0,4.5,8.0,3053.0,0.0] |43           |30.222132827927663|
|[5.0,1.0,1.0,3.5,17.0,3031.0,1.0]|36           |30.0951293511741  |
|[5.0,1.0,1.0,4.5,9.0,3141.0,0.0] |54           |32.69544711644185 |
+---------------------------------+-------------+------------------+
only showing top 5 rows

Gradient-Boosted Tree Predictions:
+---------------------------------+-------------+------------------+
|features                         |delivery_time|gbt_prediction    |
+---------------------------------+-------------+------------------+
|[5.0,0.0,1.0,0.5,11.0,3205.0,0.0]|5            |5.7869912530745955|
|[2.0,0.0,1.0,8.5,16.0,3031.0,0.0]|56           |65.79932286456994 |
|[0.0,1.0,1.0,4.5,8.0,3053.0,0.0] |43           |44.128181645123206|
|[5.0,1.0,1.0,3.5,17.0,3031.0,1.0]|36           |39.08585630731664 |
|[5.0,1.0,1.0,4.5,9.0,3141.0,0.0] |54           |51.05473224972755 |
+---------------------------------+-------------+------------------+
only showing top 5 rows
```

2.4.2 For both models (RF and GBT): with the test data, decide on which metrics to use for model evaluation and discuss which one is the better model (no word limit; please keep it concise). You may also use a plot for visualisation (not mandatory).

In [120…

```
from pyspark.ml.evaluation import RegressionEvaluator

# Define evaluators for RMSE, MAE, and R²
rmse_evaluator = RegressionEvaluator(labelCol="delivery_time", predictionCol="rf
mae_evaluator = RegressionEvaluator(labelCol="delivery_time", predictionCol="rf_
r2_evaluator = RegressionEvaluator(labelCol="delivery_time", predictionCol="rf_p

# Random Forest Model Evaluation
rf_rmse = rmse_evaluator.evaluate(rf_predictions)
rf_mae = mae_evaluator.evaluate(rf_predictions)
rf_r2 = r2_evaluator.evaluate(rf_predictions)

print(f"Random Forest Evaluation:\n RMSE: {rf_rmse:.3f}, MAE: {rf_mae:.3f}, R²:

# Update evaluator prediction column for GBT predictions
rmse_evaluator.setPredictionCol("gbt_prediction")
mae_evaluator.setPredictionCol("gbt_prediction")
r2_evaluator.setPredictionCol("gbt_prediction")
```

```
# Gradient-Boosted Tree Model Evaluation
gbt_rmse = rmse_evaluator.evaluate(gbt_predictions)
gbt_mae = mae_evaluator.evaluate(gbt_predictions)
gbt_r2 = r2_evaluator.evaluate(gbt_predictions)

print(f"Gradient-Boosted Tree Evaluation:\n RMSE: {gbt_rmse:.3f}, MAE: {gbt_mae:
```

```
Random Forest Evaluation:
 RMSE: 7.701, MAE: 5.099, R²: 0.872
Gradient-Boosted Tree Evaluation:
 RMSE: 4.413, MAE: 2.873, R²: 0.958
```

2.4.3 3. Save the better model (you'll need it for A2B). (Note: You may need to go through a few training loops or use more data to create a better-performing model.)

In [121…

```
# Define the path to save the model
best_model_path = "best_model"

# GBT is the better model based on evaluation metrics
best_model = gbt_model

# Save the model
best_model.write().overwrite().save(best_model_path)

print(f"Best model saved at: {best_model_path}")
```

```
Best model saved at: best_model
```

Part 3. Hyperparameter Tuning and Model Optimisation

Apply the techniques you have learnt from the labs, for example, CrossValidator, TrainValidationSplit, ParamGridBuilder, etc., to perform further hyperparameter tuning and model optimisation.
The assessment is based on the quality of your work/process, not the quality of your model. Please include your thoughts/ideas/discussions.

In [97]:

```
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.regression import RandomForestRegressor, GBTRegressor

# Encode categorical features using StringIndexer
categorical_features = ["weather_condition", "road_condition", "driver_type_of_v

indexers = [
    StringIndexer(inputCol=col, outputCol=f"{col}_encoded").fit(train_data)
    for col in categorical_features if col in train_data.columns
]

# Define feature columns
feature_columns = [
    "travel_distance", "weather_condition_encoded",
    "road_condition_encoded", "hour_of_day",
    "delivery_postcode", "driver_type_of_vehicle_encoded"
]

# Ensure features are assembled
```

```
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

# Define models
rf_regressor = RandomForestRegressor(featuresCol="features", labelCol="delivery_
gbt_regressor = GBTRegressor(featuresCol="features", labelCol="delivery_time", p

# Create ML Pipelines
rf_pipeline = Pipeline(stages=indexers + [assembler, rf_regressor])
gbt_pipeline = Pipeline(stages=indexers + [assembler, gbt_regressor])

# Define evaluation metric
evaluator = RegressionEvaluator(labelCol="delivery_time", predictionCol="predict

# Hyperparameter Grid for Random Forest
rf_param_grid = ParamGridBuilder() \
    .addGrid(rf_regressor.numTrees, [50, 100]) \
    .addGrid(rf_regressor.maxDepth, [5, 10]) \
    .addGrid(rf_regressor.minInstancesPerNode, [1, 2]) \
    .build()

# Hyperparameter Grid for Gradient-Boosted Tree
gbt_param_grid = ParamGridBuilder() \
    .addGrid(gbt_regressor.maxIter, [10, 20]) \
    .addGrid(gbt_regressor.maxDepth, [5, 10]) \
    .addGrid(gbt_regressor.stepSize, [0.1, 0.2]) \
    .build()

# Cross-validator for Random Forest
rf_cv = CrossValidator(
    estimator=rf_pipeline,
    estimatorParamMaps=rf_param_grid,
    evaluator=evaluator,
    numFolds=3,
    parallelism=2
)

# Cross-validator for Gradient-Boosted Tree
gbt_cv = CrossValidator(
    estimator=gbt_pipeline,
    estimatorParamMaps=gbt_param_grid,
    evaluator=evaluator,
    numFolds=3,
    parallelism=2
)

# Train models with cross-validation
rf_best_model = rf_cv.fit(train_data)
gbt_best_model = gbt_cv.fit(train_data)

# Get Predictions
rf_cv_predictions = rf_best_model.transform(test_data)
gbt_cv_predictions = gbt_best_model.transform(test_data)

# Evaluate the best models
rf_rmse = evaluator.evaluate(rf_cv_predictions)
gbt_rmse = evaluator.evaluate(gbt_cv_predictions)

print(f"Tuned Random Forest RMSE: {rf_rmse}")
print(f"Tuned Gradient-Boosted Tree RMSE: {gbt_rmse}")
```

```
Tuned Random Forest RMSE: 5.120835551104456
Tuned Gradient-Boosted Tree RMSE: 4.251499051743598
```

This PySpark ML pipeline builds and tunes Random Forest (RF) and Gradient-Boosted Tree (GBT) regressors to predict delivery time using a food delivery dataset. Categorical features (e.g., weather and road conditions) are encoded using StringIndexer, and numerical features are combined using VectorAssembler. The models undergo hyperparameter tuning via CrossValidator with 3-fold cross-validation. The best-tuned models are then evaluated using Root Mean Squared Error (RMSE). The output shows that GBT performs better (RMSE = 4.25) than RF (RMSE = 5.12), indicating that GBT predicts delivery time with lower error.

References:

Please add your references below:

In [ ]: