# 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 [328]:

```
from pyspark import SparkConf  # Import SparkConf to configure Spark applicati
from pyspark.sql import SparkSession  # Import SparkSession to manage the Spar

# Define Spark master URL and application name
master = "local[4]"  # only using 4 CPU cores
app_name = "FIT5202Ass2A"  # Name of the Spark application

# Configure SparkConf object with master URL and application name
spark_conf = SparkConf().setMaster(master).setAppName(app_name)

# Create a SparkSession with additional configuration, such as max partition si
spark = SparkSession.builder \
    .config(conf=spark_conf) \
    .config("spark.files.maxPartitionBytes", "16777216") \
    .getOrCreate() ## Set max partition size to 16MB, 16MB = 16777216 Bytes

# Get SparkContext and set log level to ERROR
sc = spark.sparkContext
sc.setLogLevel("ERROR")  # Only log error messages to reduce log output
```

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 [329]:

```
# Import required modules from PySpark
from pyspark.sql.types import *  # For defining schema and data types
import pyspark.sql.functions as F  # For performing operations on DataFrames
```

In [330]:

```
# Define the schema for delivery_address.csv
address_schema = StructType([
    StructField("gid", StringType(), True),  # ID of delivery address geolocati
    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),  # Latitude with 6 decimal pre
    StructField("longitude", DoubleType(), True),  # Geometry point on maps
    StructField("geom", StringType(), True),
    StructField("delivery_id", StringType(), True)  # ID of a delivery address
])
```

In [331]:

```
# Read the delivery_address.csv file into a DataFrame using the defined schema
df_address = spark.read.csv("./delivery_address.csv", header=True, schema=addr
```

In [332]:

```
# Define the schema for driver.csv
driver_schema = StructType([
    StructField("driver_id", StringType(), True),  # Unique identifier of deliv
    StructField("age", IntegerType(), True),  # Driver's age, range 18-60
    StructField("rating", DoubleType(), True),  # Overall rating of the driver
    StructField("year_experience", IntegerType(), True),  # Years of delivery e
    StructField("vehicle_condition", StringType(), True),  # Vehicle condition:
    StructField("type_of_vehicle", StringType(), True)  # Type of vehicle (Moto
])
```

In [333]:

```
# Read the driver.csv file into a DataFrame using the defined schema
df_driver = spark.read.csv("./driver.csv", header=True, schema=driver_schema)
```

In [334]:

```
# Define the schema for order.csv
order_schema = StructType([
    StructField("order_id", StringType(), True),  # Unique identifier of an ord
    StructField("delivery_person_id", IntegerType(), True),  # ID of the driver
    StructField("order_ts", IntegerType(), True),  # Timestamp when an order is
    StructField("ready_ts", IntegerType(), True),  # Timestamp when the order i
    StructField("weather_condition", StringType(), True),  # Weather condition
    StructField("road_condition", StringType(), True),  # Road condition during
    StructField("type_of_order", StringType(), True),  # Type of order (Snacks,
    StructField("order_total", IntegerType(), True),  # Total value of the orde
    StructField("delivery_time", IntegerType(), True),  # Delivery time excludi
    StructField("travel_distance", FloatType(), True),  # Total travel distance
    StructField("restaurant_id", StringType(), True),
    StructField("delivery_id", StringType(), True)
])
```

In [335]:

```
# Read the order.csv file into a DataFrame using the defined schema
df_order = spark.read.csv("./order.csv", header=True, schema=order_schema)
```

In [336]:

```
# Define the schema for restaurants.csv
restaurant_schema = StructType([
    StructField("row_id", IntegerType(), True),  # Row ID of the restaurant
    StructField("restaurant_code", StringType(), True),  # Internal code of a r
    StructField("chain_id", StringType(), True),  # Chain ID (empty if not part
    StructField("primary_cuisine", StringType(), True),  # Primary cuisine of t
    StructField("latitude", DoubleType(), True),  # Latitude with 6 decimal pre
    StructField("longitude", DoubleType(), True),  # Geometry point on maps
    StructField("geom", StringType(), True),  # Geometry point of the restauran
    StructField("restaurant_id", StringType(), True),  # ID of a restaurant (pr
    StructField("suburb", StringType(), True),
    StructField("postcode", IntegerType(), True)
])
```

In [337]:

```
# Read the restaurants.csv file into a DataFrame using the defined schema
df_restaurant = spark.read.csv("./restaurants.csv", header=True, schema=restau
```

In [338]:
```
# Print the schema of the delivery_address.csv DataFrame
df_address.printSchema()
```

```
root
 |-- gid: string (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: string (nullable = true)
```

In [339]:
```
# Print the schema of the driver.csv DataFrame
df_driver.printSchema()
```

```
root
 |-- driver_id: string (nullable = true)
 |-- age: integer (nullable = true)
 |-- rating: double (nullable = true)
 |-- year_experience: integer (nullable = true)
 |-- vehicle_condition: string (nullable = true)
 |-- type_of_vehicle: string (nullable = true)
```

In [340]:
```
# Print the schema of the order.csv DataFrame
df_order.printSchema()
```

```
root
 |-- order_id: string (nullable = true)
 |-- delivery_person_id: integer (nullable = true)
 |-- order_ts: integer (nullable = true)
 |-- ready_ts: integer (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: string (nullable = true)
 |-- delivery_id: string (nullable = true)
```

In [341]:
```
# Print the schema of the restaurants.csv DataFrame
df_restaurant.printSchema()
```

```
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: string (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 [342]:

```
# Import necessary modules
from pyspark.sql import SparkSession  # For creating a Spark session
from pyspark.sql.functions import col, udf  # For column operations and defini
from pyspark.sql.types import StringType, IntegerType  # For defining UDF retu

# Step 1: Initialize Spark session
# Create a Spark session for processing the dataset
spark = SparkSession.builder.appName("A2_Task").getOrCreate()

# Step 2: Define UDFs for day of the week and hour of the day
# UDF to extract the day of the week (e.g., Monday, Tuesday) from a UNIX timest
def get_day_of_week(unix_time):
    from datetime import datetime
    return datetime.utcfromtimestamp(unix_time).strftime('%A')  # Returns the 

# UDF to extract the hour of the day (0-23) from a UNIX timestamp
def get_hour_of_day(unix_time):
    from datetime import datetime
    return datetime.utcfromtimestamp(unix_time).hour  # Returns the hour of the

# Register UDFs for use with DataFrame columns
get_day_of_week_udf = udf(get_day_of_week, StringType())
get_hour_of_day_udf = udf(get_hour_of_day, IntegerType())

# Step 3: Load dataset
# Load the order.csv file into a DataFrame with headers and infer schema types
df_order = spark.read.csv("./order.csv", header=True, inferSchema=True)

# Step 4: Ensure order_ts is in the correct format
# Convert the order_ts column to long type to ensure it is a valid UNIX timesta
df_order = df_order.withColumn("order_ts", col("order_ts").cast("long"))

# Step 5: Add day_of_week and hour_of_day columns
# Use UDFs to create two new columns: day_of_week and hour_of_day
# Select only the required columns: order_id, day_of_week, hour_of_day
feature_df = df_order \
    .withColumn("day_of_week", get_day_of_week_udf(col("order_ts"))) \
    .withColumn("hour_of_day", get_hour_of_day_udf(col("order_ts"))) \
    .select("order_id", "delivery_id", "day_of_week", "hour_of_day")  # Keep on

# Step 6: Display the result
# Show the first 10 rows of the transformed DataFrame with all columns visible
feature_df.show(10, truncate=False)
```

| order_id | delivery_id | day_of_week | hour_of_day |
|---|---|---|---|
| 02bccb12-7bb2-41c0-af35-3fe34f6e48f7 | 7530 | Monday | 20 |
| c805e0fd-2214-4dc6-b4bd-ef93bfc63d33 | 7355 | Wednesday | 21 |
| 5aba5eac-ab01-4bfa-9805-2cf34a52109e | 9140 | Tuesday | 5 |
| f258e133-bea0-46b3-80eb-13de47ff1325 | 23 | Wednesday | 10 |
| b8955ebc-2e67-4a9d-b49f-b56ba6cdcf7e | 1765 | Wednesday | 11 |
| 500cd68e-b7bb-4af4-8748-8140659183f5 | 8720 | Saturday | 21 |
| 8b96a6c9-34d2-4fc2-9401-ab86a1b5a977 | 6536 | Sunday | 13 |
| 3b52cfa9-8960-4406-93e0-a3489b7cc2ce | 8818 | Saturday | 20 |
| 8a3f6783-dfd7-4591-b7ec-764bee1ce97f | 6074 | Friday | 21 |
| 72c06040-5442-4dd7-ac2f-310c9a3462ca | 4594 | Sunday | 5 |

only showing top 10 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 [419]:

```
# Step 1: Define a UDF for detecting peak hours
# This UDF checks if the input hour (0-23) is within the peak hour range
# Peak hours are defined as [7, 8, 9, 16, 17, 18]
def is_peak_hour(hour):
    peak_hours = [7, 8, 9, 16, 17, 18]  # Define peak hours
    return hour in peak_hours  # Return True if the hour is a peak hour, False

# Register the UDF
is_peak_hour_udf = udf(is_peak_hour, StringType())  # UDF returns 'true' or 'fa

# Step 2: Add a new column to the DataFrame to identify peak hours
# Use the `withColumn` function and apply the UDF to the hour_of_day column
feature_df_with_peak = feature_df \
    .withColumn("is_peak", is_peak_hour_udf(col("hour_of_day"))) \
    .select("order_id", "delivery_id", "day_of_week", "hour_of_day", "is_peak")

# Step 3: Display the resulting DataFrame
# Show the first 10 rows of the transformed DataFrame
feature_df_with_peak.show(10, truncate=False)
```

```
+------------------------------------+-----------+-----------+-----------+-------
+
|order_id                            |delivery_id|day_of_week|hour_of_day|is_peak
|
+------------------------------------+-----------+-----------+-----------+-------
+
|02bccb12-7bb2-41c0-af35-3fe34f6e48f7|7530       |0          |20         |false
|
|c805e0fd-2214-4dc6-b4bd-ef93bfc63d33|7355       |2          |21         |false
|
|5aba5eac-ab01-4bfa-9805-2cf34a52109e|9140       |1          |5          |false
|
|f258e133-bea0-46b3-80eb-13de47ff1325|23         |2          |10         |false
|
|b8955ebc-2e67-4a9d-b49f-b56ba6cdcf7e|1765       |2          |11         |false
|
|500cd68e-b7bb-4af4-8748-8140659183f5|8720       |5          |21         |false
|
|8b96a6c9-34d2-4fc2-9401-ab86a1b5a977|6536       |6          |13         |false
|
|3b52cfa9-8960-4406-93e0-a3489b7cc2ce|8818       |5          |20         |false
|
|8a3f6783-dfd7-4591-b7ec-764bee1ce97f|6074       |4          |21         |false
|
|72c06040-5442-4dd7-ac2f-310c9a3462ca|4594       |6          |5          |false
|
+------------------------------------+-----------+-----------+-----------+-------
+
only showing top 10 rows
```

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

In [420]:

```
# Rename delivery address and restaurant DataFrame columns to avoid conflicts du
df_address_renamed = df_address \
    .withColumnRenamed("delivery_id", "address_delivery_id") \
    .withColumnRenamed("suburb", "address_suburb") \
    .withColumnRenamed("postcode", "address_postcode") \
    .withColumnRenamed("latitude", "address_latitude") \
    .withColumnRenamed("longitude", "address_longitude") \
    .withColumnRenamed("geom", "address_geom")

df_restaurant_renamed = df_restaurant \
    .withColumnRenamed("suburb", "restaurant_suburb") \
    .withColumnRenamed("postcode", "restaurant_postcode") \
    .withColumnRenamed("latitude", "restaurant_latitude") \
    .withColumnRenamed("longitude", "restaurant_longitude") \
    .withColumnRenamed("geom", "restaurant_geom")\
    .withColumnRenamed("restaurant_id", "restaurant_restaurant_id")

df_order_renamed = df_order \
    .withColumnRenamed("delivery_id", "order_delivery_id")\
    .withColumnRenamed("restaurant_id", "order_restaurant_id")
```

In [421]:

```
from pyspark.sql.functions import col

# Join delivery address data with orders using order_delivery_id
df_with_delivery_suburb = df_order_renamed.join(
    df_address_renamed,
    col("order_delivery_id") == col("address_delivery_id"),
    how="left"
).drop("address_delivery_id")  # Remove duplicate column

# Join restaurant data with orders using order_restaurant_id
df_with_geolocation = df_with_delivery_suburb.join(
    df_restaurant_renamed,
    col("order_restaurant_id") == col("restaurant_restaurant_id"),
    how="left"
).drop("restaurant_restaurant_id")  # Remove duplicate column

# Select required columns and show the result
df_with_geolocation.select("order_id", "address_suburb", "restaurant_suburb").sl
```

```
+------------------------------------+---------------+-----------------+
|order_id                            |address_suburb |restaurant_suburb|
+------------------------------------+---------------+-----------------+
|02bccb12-7bb2-41c0-af35-3fe34f6e48f7|SOUTH YARRA    |EAST MELBOURNE   |
|c805e0fd-2214-4dc6-b4bd-ef93bfc63d33|PRAHRAN        |KENSINGTON       |
|5aba5eac-ab01-4bfa-9805-2cf34a52109e|PORT MELBOURNE |PORT MELBOURNE   |
|f258e133-bea0-46b3-80eb-13de47ff1325|MELBOURNE      |PARKVILLE        |
|b8955ebc-2e67-4a9d-b49f-b56ba6cdcf7e|MELBOURNE      |CARLTON          |
|500cd68e-b7bb-4af4-8748-8140659183f5|NORTH MELBOURNE|PORT MELBOURNE   |
|8b96a6c9-34d2-4fc2-9401-ab86a1b5a977|WEST MELBOURNE |SOUTH MELBOURNE  |
|3b52cfa9-8960-4406-93e0-a3489b7cc2ce|SOUTH YARRA    |EAST MELBOURNE   |
|8a3f6783-dfd7-4591-b7ec-764bee1ce97f|SOUTH YARRA    |SOUTH YARRA      |
|72c06040-5442-4dd7-ac2f-310c9a3462ca|DOCKLANDS      |NORTH MELBOURNE  |
+------------------------------------+---------------+-----------------+
only showing top 10 rows
```

1.2.4 Join data frames to add restaurant information to the feature_df: primary_cuisine,

In [433]:
```
# Alias the dataframes to avoid column ambiguity
df_order_alias = feature_df.alias("orders")
df_restaurant_alias = df_restaurant.alias("restaurants")

# Join restaurant information to feature_df
feature_df = df_order_alias.join(
    df_restaurant_alias.select(
        col("restaurants.restaurant_id"),
        col("restaurants.primary_cuisine"),
        col("restaurants.latitude"),
        col("restaurants.longitude"),
        col("restaurants.suburb"),
        col("restaurants.postcode")
    ),
    col("orders.delivery_id") == col("restaurants.restaurant_id"),
    how="left"
)

# Drop the duplicate restaurant_id column after join
feature_df = feature_df.drop("restaurant_id")

# Print the first 5 rows of the updated feature_df
print("Updated feature_df with restaurant information:")
feature_df.show(5, truncate=False)
```

```
Updated feature_df with restaurant information:
+------------------------------------+-----------+-----------+-----------+-------
--------+------------+------------+--------------+--------+---------------+------
------+------------+--------------+--------+---------------+------------+--------
----+--------------+--------+---------------+------------+------------+----------
----+--------+---------------+------------+------------+--------------+--------+-
--------------+------------+------------+--------------+--------+---------------+
------------+------------+--------------+--------+---------------+------------+--
----------+--------------+--------+---------------+------------+------------+----
----------+--------+---------------+------------+------------+--------------+----
----+
|order_id                            |delivery_id|day_of_week|hour_of_day|primary
_cuisine|latitude    |longitude   |suburb        |postcode|primary_cuisine|latitu
de    |longitude   |suburb        |postcode|primary_cuisine|latitude    |longitud
e   |suburb        |postcode|primary_cuisine|latitude    |longitude   |suburb
|postcode|primary_cuisine|latitude    |longitude   |suburb        |postcode|prima
ry_cuisine|latitude    |longitude   |suburb        |postcode|primary_cuisine|lati
tude    |longitude   |suburb        |postcode|primary_cuisine|latitude    |longit
ude   |suburb        |postcode|primary_cuisine|latitude    |longitude   |suburb
|postcode|primary_cuisine|latitude    |longitude   |suburb        |postcode|
+------------------------------------+-----------+-----------+-----------+-------
--------+------------+------------+--------------+--------+---------------+------
------+------------+--------------+--------+---------------+------------+--------
----+--------------+--------+---------------+------------+------------+----------
----+--------+---------------+------------+------------+--------------+--------+-
--------------+------------+------------+--------------+--------+---------------+
------------+------------+--------------+--------+---------------+------------+--
----------+--------------+--------+---------------+------------+------------+----
----------+--------+---------------+------------+------------+--------------+----
----+
|02bccb12-7bb2-41c0-af35-3fe34f6e48f7|7530       |0          |20         |NULL
|NULL        |NULL        |NULL          |NULL    |NULL           |NULL        |N
ULL        |NULL          |NULL    |NULL           |NULL        |NULL        |NUL
L          |NULL    |NULL           |NULL        |NULL        |NULL          |NUL
L    |NULL           |NULL        |NULL        |NULL          |NULL    |NULL
|NULL        |NULL        |NULL          |NULL    |NULL           |NULL        |N
ULL        |NULL          |NULL    |NULL           |NULL        |NULL        |NUL
L          |NULL    |NULL           |NULL        |NULL        |NULL          |NUL
L    |NULL           |NULL        |NULL        |NULL          |NULL    |
|c805e0fd-2214-4dc6-b4bd-ef93bfc63d33|7355       |2          |21         |NULL
|NULL        |NULL        |NULL          |NULL    |NULL           |NULL        |N
ULL        |NULL          |NULL    |NULL           |NULL        |NULL        |NUL
L          |NULL    |NULL           |NULL        |NULL        |NULL          |NUL
L    |NULL           |NULL        |NULL        |NULL          |NULL    |NULL
|NULL        |NULL        |NULL          |NULL    |NULL           |NULL        |N
ULL        |NULL          |NULL    |NULL           |NULL        |NULL        |NUL
L          |NULL    |NULL           |NULL        |NULL        |NULL          |NUL
L    |NULL           |NULL        |NULL        |NULL          |NULL    |
|5aba5eac-ab01-4bfa-9805-2cf34a52109e|9140       |1          |5          |NULL
|NULL        |NULL        |NULL          |NULL    |NULL           |NULL        |N
ULL        |NULL          |NULL    |NULL           |NULL        |NULL        |NUL
L          |NULL    |NULL           |NULL        |NULL        |NULL          |NUL
L    |NULL           |NULL        |NULL        |NULL          |NULL    |NULL
|NULL        |NULL        |NULL          |NULL    |NULL           |NULL        |N
ULL        |NULL          |NULL    |NULL           |NULL        |NULL        |NUL
L          |NULL    |NULL           |NULL        |NULL        |NULL          |NUL
L    |NULL           |NULL        |NULL        |NULL          |NULL    |
|f258e133-bea0-46b3-80eb-13de47ff1325|23         |2          |10         |Beverag
es      |-37.83666008|144.94078568|PORT MELBOURNE|3207    |Beverages      |-37.83
666008|144.94078568|PORT MELBOURNE|3207    |Beverages      |-37.83666008|144.9407
8568|PORT MELBOURNE|3207    |Beverages      |-37.83666008|144.94078568|PORT MELBO
```

```
URNE|3207     |Beverages       |-37.83666008|144.94078568|PORT MELBOURNE|3207     |B
everages       |-37.83666008|144.94078568|PORT MELBOURNE|3207     |Beverages       |
-37.83666008|144.94078568|PORT MELBOURNE|3207     |Beverages       |-37.83666008|14
4.94078568|PORT MELBOURNE|3207     |Beverages       |-37.83666008|144.94078568|PORT
MELBOURNE|3207     |Beverages       |-37.83666008|144.94078568|PORT MELBOURNE|3207
|
|b8955ebc-2e67-4a9d-b49f-b56ba6cdcf7e|1765       |2           |11          |NULL
|NULL         |NULL         |NULL           |NULL     |NULL           |NULL         |N
ULL         |NULL           |NULL     |NULL           |NULL         |NULL         |NUL
L           |NULL    |NULL           |NULL         |NULL         |NULL           |NUL
L     |NULL            |NULL         |NULL         |NULL           |NULL     |NULL
|NULL         |NULL         |NULL           |NULL     |NULL           |NULL         |N
ULL         |NULL           |NULL     |NULL           |NULL         |NULL         |NUL
L           |NULL    |NULL           |NULL         |NULL         |NULL           |NUL
L     |NULL            |NULL         |NULL         |NULL           |NULL     |
+------------------------------------+------------+------------+------------+-------
--------+------------+------------+---------------+---------+---------------+------
------+------------+---------------+---------+----------------+------------+--------
----+---------------+---------+----------------+------------+------------+----------
----+---------+----------------+------------+------------+---------------+--------+-
--------------+------------+------------+---------------+---------+---------------+
------------+------------+---------------+---------+----------------+------------+--
----------+---------------+---------+----------------+------------+------------+----
----------+---------+----------------+------------+------------+---------------+----
----+
only showing top 5 rows
```

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 [434]:

```
# Step 1: Rename columns in driver.csv to avoid conflicts
df_driver_renamed = df_driver \
    .withColumnRenamed("driver_id", "driver_driver_id")

# Step 2: Ensure data types match between join columns
df_order_renamed = df_order \
    .withColumnRenamed("delivery_person_id", "order_delivery_person_id") \
    .withColumn("order_delivery_person_id", col("order_delivery_person_id").cas

df_driver_renamed = df_driver_renamed \
    .withColumn("driver_driver_id", col("driver_driver_id").cast("string"))

# Step 3: Perform the join to add type_of_vehicle to order DataFrame
df_order_with_vehicle = df_order_renamed.join(
    df_driver_renamed.select("driver_driver_id", "type_of_vehicle"),
    df_order_renamed["order_delivery_person_id"] == df_driver_renamed["driver_d
    how="left"  # Use left join to retain all orders even if no matching driver
)

# Step 4: Select relevant columns, including the new type_of_vehicle column
result_df = df_order_with_vehicle.select(
    "order_id",  # Keep all columns from the original order DataFrame
    "order_delivery_person_id",
    "type_of_vehicle",
    "order_ts",
    "delivery_time",
    "order_total"
)

# Step 5: Display the schema and data
result_df.show(10, truncate=False)
```

```
+------------------------------------+------------------------+----------------+--
--------+-------------+-----------+
|order_id                            |order_delivery_person_id|type_of_vehicle|or
der_ts  |delivery_time|order_total|
+------------------------------------+------------------------+----------------+--
--------+-------------+-----------+
|02bccb12-7bb2-41c0-af35-3fe34f6e48f7|1313                    |Car            |17
33172480|3            |13         |
|c805e0fd-2214-4dc6-b4bd-ef93bfc63d33|1589                    |Scooter        |17
12178816|7            |80         |
|5aba5eac-ab01-4bfa-9805-2cf34a52109e|1554                    |Motorcycle     |17
21109376|30           |20         |
|f258e133-bea0-46b3-80eb-13de47ff1325|1520                    |eBike          |17
13955200|29           |5          |
|b8955ebc-2e67-4a9d-b49f-b56ba6cdcf7e|1763                    |Bike           |17
10328448|4            |202        |
|500cd68e-b7bb-4af4-8748-8140659183f5|1625                    |Bike           |17
11230720|24           |17         |
|8b96a6c9-34d2-4fc2-9401-ab86a1b5a977|1751                    |Car            |17
25801216|9            |14         |
|3b52cfa9-8960-4406-93e0-a3489b7cc2ce|1866                    |Motorcycle     |17
15460736|11           |21         |
|8a3f6783-dfd7-4591-b7ec-764bee1ce97f|1511                    |Bike           |17
11142912|71           |7          |
|72c06040-5442-4dd7-ac2f-310c9a3462ca|1703                    |Motorcycle     |17
10653440|3            |12         |
+------------------------------------+------------------------+----------------+--
--------+-------------+-----------+
only showing top 10 rows
```

## 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 [435]:

```
from IPython.display import display

# Numeric summary
print("Numeric column statistics:")
display(numeric_summary.style.set_table_attributes("style='display:inline'").se

# Categorical columns
print("\nTop 5 values for categorical columns:")
for col in categorical_cols:
    top_5 = (
        feature_df.groupBy(col)
        .count()
        .orderBy(F.col("count").desc())
        .limit(5)
        .toPandas()
    )
    print(f"\nColumn: {col}")
    display(top_5.style.set_table_attributes("style='display:inline'").set_capt

# Boolean columns
print("\nValue counts for boolean columns:")
if boolean_cols:
    for col in boolean_cols:
        value_counts = (
            feature_df.groupBy(col).count().orderBy(F.col(col)).toPandas()
        )
        print(f"\nColumn: {col}")
        display(value_counts.style.set_table_attributes("style='display:inline'
else:
    print("No boolean columns found.")
```

```
Numeric column statistics:
```

Numeric Summary

| | summary | order_id | delivery_id |
|---|---|---|---|
| **0** | count | 3 | 3 |
| **1** | mean | 2.0 | 2.0 |
| **2** | stddev | 1.0 | 1.0 |
| **3** | min | 1 | 1 |
| **4** | max | 3 | 3 |

```
Top 5 values for categorical columns:

Column: day_of_week
```

Top 5 Values for
day_of_week

| | day_of_week | count |
|---|---|---|
| **0** | 6 | 137504 |
| **1** | 0 | 137276 |
| **2** | 1 | 136344 |
| **3** | 5 | 135734 |
| **4** | 3 | 134271 |

```
Column: hour_of_day
```

Top 5 Values for
hour_of_day

| | hour_of_day | count |
|---|---|---|
| **0** | 0 | 40991 |
| **1** | 8 | 40902 |
| **2** | 16 | 40673 |
| **3** | 12 | 39655 |
| **4** | 14 | 39619 |

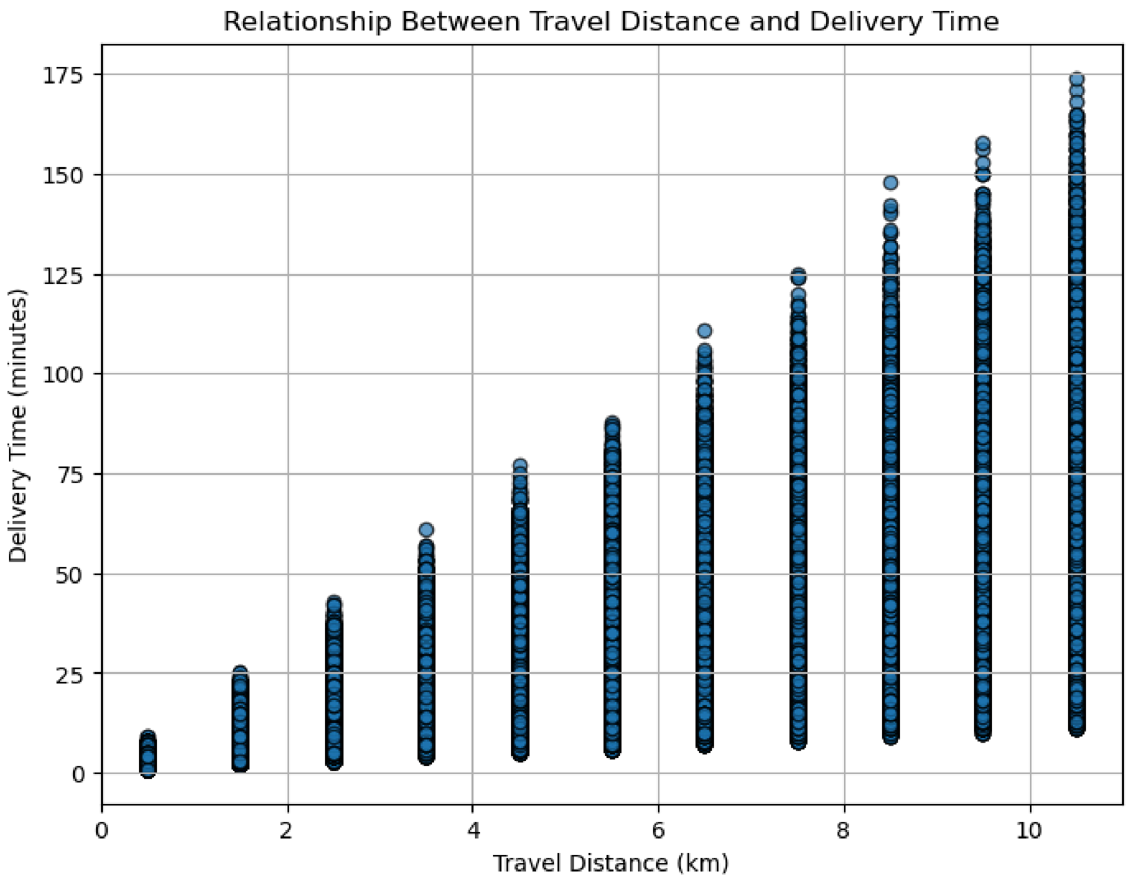
```
Value counts for boolean columns:
No boolean columns found.
```

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 [436]:

```
import pandas as pd
import matplotlib.pyplot as plt

# read the order.csv
file_path = './order.csv'
df = pd.read_csv(file_path)

# Select the desired column and remove the missing values
df = df[['travel_distance', 'delivery_time']].dropna()

# Draw the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(df['travel_distance'], df['delivery_time'], alpha=0.7, edgecolor='k
plt.title('Relationship Between Travel Distance and Delivery Time')
plt.xlabel('Travel Distance (km)')
plt.ylabel('Delivery Time (minutes)')
plt.grid(True)
plt.show()

# 150 words analysis
print("""
Analysis of Scatter Plot:
The scatter plot shows a positive correlation between travel distance and deliv
As the travel distance increases, the delivery time also tends to increase, whi
However, further analysis is required to account for other factors such as traf
""")
```

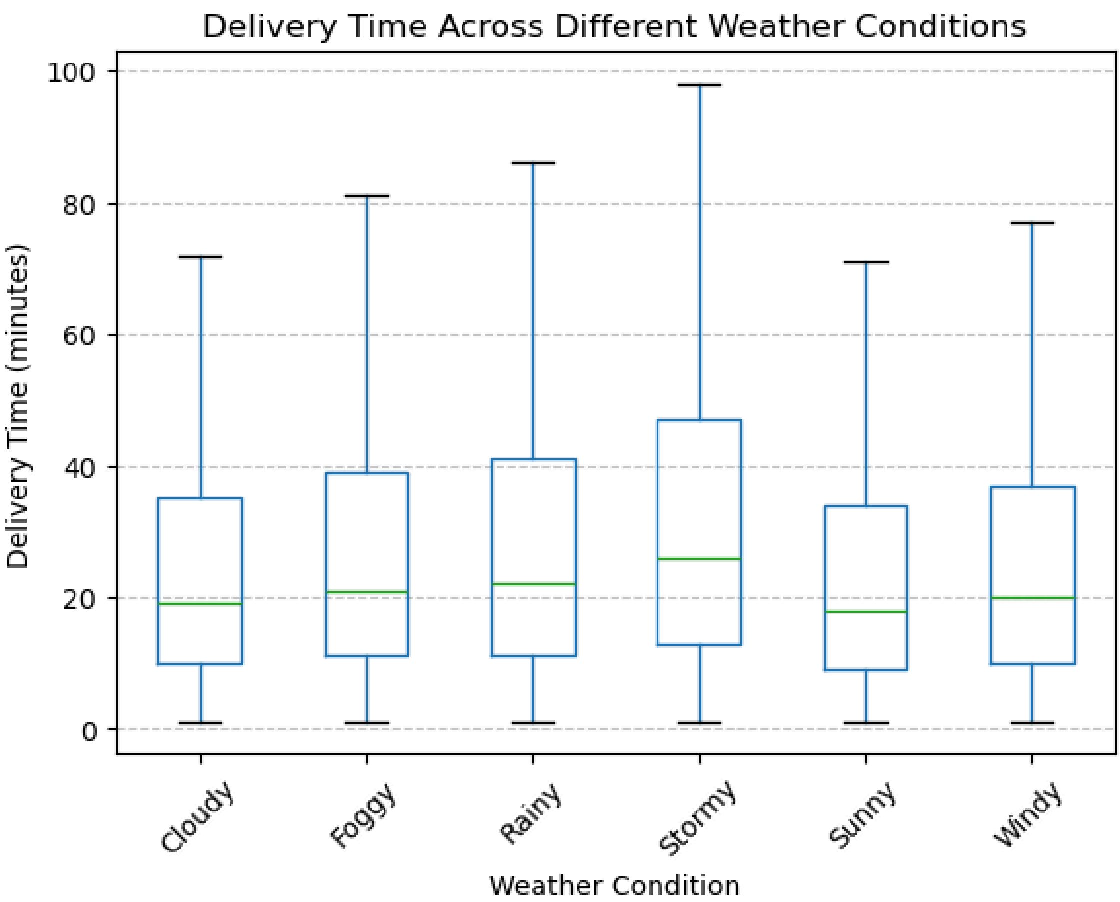


```
Analysis of Scatter Plot:
The scatter plot shows a positive correlation between travel distance and deliver
y time.
As the travel distance increases, the delivery time also tends to increase, which
aligns with expectations.
However, further analysis is required to account for other factors such as traffi
c and weather conditions that may also influence delivery time.
```

In [437]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the order.csv
file_path = './order.csv'
df = pd.read_csv(file_path)

# Select the desired column and remove the missing values
df = df[['weather_condition', 'delivery_time']].dropna()

# Draw the box plot
plt.figure(figsize=(10, 6))
df.boxplot(column='delivery_time', by='weather_condition', grid=False, showfli
plt.title('Delivery Time Across Different Weather Conditions')
plt.suptitle('')
plt.xlabel('Weather Condition')
plt.ylabel('Delivery Time (minutes)')
plt.xticks(rotation=45)
plt.grid(True, axis='y', linestyle='--', alpha=0.7)
plt.show()

# 150 words analysis
print("""
Analysis of Box Plot:
The box plot shows the distribution of delivery times under different weather c
From the plot, we can observe how extreme weather (e.g., Rainy or Stormy) might
the delivery time due to safety precautions and slower travel speeds.
Sunny and Windy conditions generally exhibit shorter delivery times. The findin
that weather conditions significantly influence delivery efficiency.
""")
```

<Figure size 1000x600 with 0 Axes>



```
Analysis of Box Plot:
The box plot shows the distribution of delivery times under different weather con
ditions.
From the plot, we can observe how extreme weather (e.g., Rainy or Stormy) might i
ncrease
the delivery time due to safety precautions and slower travel speeds.
Sunny and Windy conditions generally exhibit shorter delivery times. The findings
emphasize
that weather conditions significantly influence delivery efficiency.
```

# 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

```
Predicting delivery time effectively requires selecting features that have a direct impact on the outcome and removing those that do not contribute to the prediction. Columns like order_id, delivery_id, and delivery_person_id are random identifiers unique to each record and do not influence the delivery time prediction. These columns can be safely removed, as they introduce unnecessary noise without adding any predictive value.
Among the most important features to retain and enhance are travel_distance, weather_conditions, and vehicle_type. Travel distance, calculated using the Euclidean distance between the latitude and longitude of the restaurant and the delivery address, directly reflects the spatial distance involved in the delivery. Weather conditions can significantly affect delivery times, as adverse weather (e.g., rain or storm) might delay deliveries due to safety precautions and slower travel speeds. Vehicle type is another critical feature, as different vehicles (e.g., motorcycles vs. cars) have varying efficiencies in navigating traffic, which can influence delivery times.
To further enhance the dataset, we calculate processing_time as the difference between ready_ts and order_ts, providing insight into the time taken to prepare the order. We also create a weather_score by mapping different weather conditions to numerical values, capturing the potential impact of weather on delivery times. Using StringIndexer, we encode categorical features like type_of_order to numerical values, making them suitable for machine learning algorithms.
```

By focusing on these core predictors and removing unnecessary columns, we aim to build a robust and interpretable model. This approach ensures that the data preparation process prioritizes clarity and relevance, enabling the model to deliver accurate predictions efficiently. The use of encoding for categorical features and the calculated processing_time allows the model to leverage all relevant information effectively. By minimizing the dataset's complexity and focusing on essential predictors, we ensure the model's robustness and interpretability, leading to efficient and accurate delivery time predictions.

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

In [467]:

```
from pyspark.ml.feature import StringIndexer

# Define the new feature DataFrame
feature_df = df_order_renamed \
    .join(
        df_restaurant_renamed.select("restaurant_restaurant_id", "restaurant_la
        df_order_renamed["order_restaurant_id"] == df_restaurant_renamed["resta
        "left"
    ) \
    .join(
        df_address_renamed.select("address_delivery_id", "address_latitude", "a
        df_order_renamed["order_restaurant_id"] == df_address_renamed["address_
        "left"
    )

# Calculate order processing time
feature_df = feature_df.withColumn(
    "processing_time",
    F.col("ready_ts") - F.col("order_ts")
)

# Create weather score
weather_score_mapping = {
    "Sunny": 5,
    "Cloudy": 4,
    "Rainy": 3,
    "Windy": 2,
    "Stormy": 1,
    "Foggy": 2
}
weather_score_udf = F.udf(lambda weather: weather_score_mapping.get(weather, 0)
feature_df = feature_df.withColumn(
    "weather_score",
    weather_score_udf(F.col("weather_condition"))
)

# Encode order type
indexer = StringIndexer(inputCol="type_of_order", outputCol="order_type_encoded
feature_df = indexer.fit(feature_df).transform(feature_df)

# Select and display feature columns
feature_df = feature_df.select(
    "order_total",          # Order total amount
    "processing_time",      # Order processing time
    "weather_score",        # Weather score
    "order_type_encoded",   # Encoded order type
    "travel_distance"       # Existing travel distance feature
)

# Display the updated feature DataFrame
feature_df.show(truncate=False)
```

```
+-----------+---------------+-------------+------------------+---------------+
|order_total|processing_time|weather_score|order_type_encoded|travel_distance|
+-----------+---------------+-------------+------------------+---------------+
|13         |128            |3            |0.0               |1.5            |
|80         |256            |4            |2.0               |1.5            |
|20         |128            |1            |3.0               |10.5           |
|5          |0              |2            |3.0               |8.5            |
|202        |256            |4            |4.0               |0.5            |
|17         |128            |2            |0.0               |2.5            |
|14         |512            |2            |1.0               |4.5            |
|21         |256            |2            |2.0               |7.5            |
|7          |256            |3            |3.0               |10.5           |
|12         |128            |1            |2.0               |1.5            |
|17         |0              |3            |0.0               |4.5            |
|20         |640            |1            |3.0               |3.5            |
|18         |128            |2            |0.0               |5.5            |
|212        |1024           |3            |4.0               |2.5            |
|179        |512            |1            |2.0               |1.5            |
|497        |256            |3            |4.0               |10.5           |
|38         |256            |2            |4.0               |5.5            |
|287        |256            |5            |4.0               |8.5            |
|14         |0              |1            |1.0               |4.5            |
|15         |256            |4            |3.0               |3.5            |
+-----------+---------------+-------------+------------------+---------------+
only showing top 20 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 [499]:

```
1
```

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

In [504]:

```
1
```

```
Columns in the DataFrame:
['order_total', 'processing_time', 'weather_score', 'order_type_encoded', 'travel
_distance']
```

## 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 [505]:
```
# 随机采样20%的数据
sampled_df = feature_df.sample(False, 0.2, seed=2025)

# 划分训练集和测试集，使用2025作为随机种子
train, test = sampled_df.randomSplit([0.8, 0.2], seed=2025)
```

### 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 [ ]:
```

```

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 [ ]:
```

```

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 [ ]:
```

```

### 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 [ ]:
```

```

Type *Markdown* and LaTeX: $\alpha^2$

## References:

Please add your references below:

In [ ]:
```

```

In [ ]:
```

```